UM1884
User manual

Description of STM32L4/L4+ HAL and low-layer drivers

Introduction

STMCube™ is STMicroelectronics's original initiative to ease developers' life by reducing development efforts, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL4 for STM32L4 series and STM32L4+ series)
  – The STM32Cube Hardware Abstraction Layer (HAL), an STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio. The HAL is available for all peripherals.
  – The low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. The LL APIs are available only for a set of peripherals.
  – A consistent set of middleware components such as RTOS, USB, Graphics
  – All embedded software utilities coming with a full set of examples.

The HAL driver layer provides a generic multi-instance simple set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks).

The HAL driver APIs are split into two categories: generic APIs which provide common and generic functions for all the STM32 series and extension APIs which include specific and customized functions for a given line or part number. The HAL drivers include a complete set of ready-to-use APIs which simplify the user application implementation. As an example, the communication peripherals contain APIs to initialize and configure the peripheral, manage data transfers in polling mode, handle interrupts or DMA, and manage communication errors.

The HAL drivers are feature-oriented instead of IP-oriented. As an example, the timer APIs are split into several categories following the IP functions: basic timer, capture, pulse width modulation (PWM), etc..

The HAL driver layer implements run-time failure detection by checking the input values of all functions. Such dynamic checking contributes to enhance the firmware robustness. Run-time detection is also suitable for user application development and debugging.

The LL drivers offer hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide atomic operations that must be called following the programming model described in the product line reference manual. As a result, the LL services are not based on standalone processes and do not require any additional memory resources to save their states, counter or data pointers: all operations are performed by changing the associated peripheral registers content. Contrary to the HAL, the LL APIs are not provided for peripherals for which optimized access is not a key feature, or for those requiring heavy software configuration and/or complex upper level stack (such as FSMC, USB, SDMMC).

The HAL and LL are complementary and cover a wide range of applications requirements:

- The HAL offers high-level and feature-oriented APIs, with a high-portability level. They hide the MCU and peripheral complexity to end-user.
- The LL offers low-level APIs at registers level, with better optimization but less portability. They require deep knowledge of the MCU and peripherals specifications.

The source code of HAL and LL drivers is developed in Strict ANSI-C which makes it independent from the development tools. It is checked with CodeSonar™ static analysis tool. It is fully documented and is MISRA-C 2004 compliant.

Contents

List of tables

List of figures

1 Acronyms and definitions

Table 1: Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
AES	Advanced encryption standard
ANSI	American National Standards Institute
API	Application Programming Interface
BSP	Board Support Package
CAN	Controller area network
CMSIS	Cortex Microcontroller Software Interface Standard
COMP	Comparator
CPU	Central Processing Unit
CRC	CRC calculation unit
CSS	Clock security system
DAC	Digital to analog converter
DFSDM	Digital filter sigma delta modulator
DMA	Direct Memory Access
EXTI	External interrupt/event controller
FLASH	Flash memory
FMC	Flexible memory controller
FW	Firewall
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
HCD	USB Host Controller Driver
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LCD	Liquid Crystal Display Controler
LPTIM	Low-power timer
LPUART	Low-power universal asynchronous receiver/transmitter
MCO	Microcontroller clock output
MPU	Memory protection unit
MSP	MCU Specific Package
NAND	NAND Flash memory
NOR	Nor Flash memory
NVIC	Nested Vectored Interrupt Controller

Acronym	Definition
OPAMP	Operational amplifier
OTG-FS	USB on-the-go full-speed
PCD	USB Peripheral Controller Driver
PWR	Power controller
QSPI	QuadSPI Flash memory
RCC	Reset and clock controller
RNG	Random number generator
RTC	Real-time clock
SAI	Serial audio interface
SD	Secure Digital
SDMMC	SD/SDIO/MultiMediaCard card host interface
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface
SWPMI	Serial Wire Protocol master interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
TSC	Touch sensing controller
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

2 Overview of HAL drivers

The HAL drivers are designed to offer a rich set of APIs and to interact easily with the application upper layers.

Each driver consists of a set of functions covering the most common peripheral features. The development of each driver is driven by a common API which standardizes the driver structure, the functions and the parameter names.

The HAL drivers include a set of driver modules, each module being linked to a standalone peripheral. However, in some cases, the module is linked to a peripheral functional mode. As an example, several modules exist for the USART peripheral: UART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode.
- Support of peripheral multi-instance allowing concurrent API calls for multiple instances of a given peripheral (USART1, USART2...)
- All HAL APIs implement user-callback functions mechanism:
  - Peripheral Init/DeInit HAL APIs can call user-callback functions to perform peripheral system level Initialization/De-Initialization (clock, GPIOs, interrupt, DMA)
  - Peripherals interrupt events
  - Error events.
- Object locking mechanism: safe hardware access to prevent multiple spurious accesses to shared resources.
- Timeout used for all blocking processes: the timeout can be a simple counter or a timebase.

2.1 HAL and user-application files

2.1.1 HAL driver files

A HAL drivers are composed of the following set of files:

Table 2: HAL driver files

File	Description
stm32l4xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32l4xx_hal_adc.c, stm32l4xx_hal_irda.c, …*
stm32l4xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example: stm32l4xx_hal_adc.h, stm32l4xx_hal_irda.h, …*

File	Description
stm32l4xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example: stm32l4xx_hal_adc_ex.c, stm32l4xx_hal_flash_ex.c, …*
stm32l4xx_hal_ppp_ex.h	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32l4xx_hal_adc_ex.h, stm32l4xx_hal_flash_ex.h, …*
stm32l4xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32l4xx_hal.h	*stm32l4*xx_hal.c header file
stm32l4xx_hal_msp_template.c	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32l4xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application.
stm32l4xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros.

2.1.2 User-application files

The minimum files required to build an application using the HAL are listed in the table below:

Table 3: User-application files

File	Description
system_stm32l4xx.c	This file contains SystemInit() which is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows relocating the vector table in internal SRAM.
startup_stm32l4xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
stm32l4xx_flash.icf (optional)	Linker file for EWARM toolchain allowing mainly adapting the stack/heap size to fit the application requirements.
stm32l4xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32l4xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.

File	Description
stm32l4xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32l4xx*_hal.c) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
main.c/.h	This file contains the main program routine, mainly: • the call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

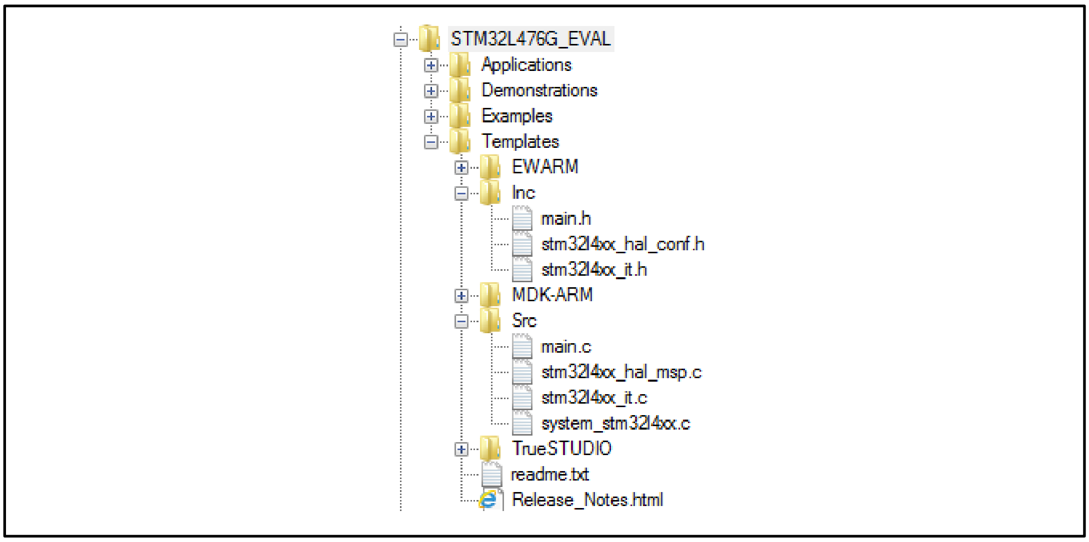
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Its features are the following:

- It contains the sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows configuring the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
  - HAL is initialized
  - SysTick ISR implemented for HAL_GetTick()
  - System clock configured with the selected device frequency.

If an existing project is copied to another location, then include paths must be updated.

Figure 1: Example of project template



2.2 HAL data structures

Each HAL driver can contain the following data structures:

- Peripheral handle structures
- Initialization and configuration structures
- Specific process structures.

2.2.1 Peripheral handle structures

The APIs have a modular generic multi-instance architecture that allows working with several IP instances simultaneously.

PPP_HandleTypeDef *handle is the main structure that is implemented in the HAL drivers. It handles the peripheral/module configuration and registers and embeds all the structures and variables needed to follow the peripheral device flow.

The peripheral handle is used for the following purposes:

- Multi instance support: each peripheral/module instance has its own handle. As a result instance resources are independent.
- Peripheral process intercommunication: the handle is used to manage shared data resources between the process routines.
  Example: global pointers, DMA handles, state machine.
- Storage: this handle is used also to manage global variables within a given HAL driver.

An example of peripheral structure is shown below:

```
typedef struct
{
USART_TypeDef *Instance; /* USART registers base address */
USART_InitTypeDef Init; /* Usart communication parameters */
uint8_t *pTxBuffPtr;/* Pointer to Usart Tx transfer Buffer */
uint16_t TxXferSize; /* Usart Tx Transfer size */
__IO uint16_t TxXferCount;/* Usart Tx Transfer Counter */
uint8_t *pRxBuffPtr;/* Pointer to Usart Rx transfer Buffer */
uint16_t RxXferSize; /* Usart Rx Transfer size */
__IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
HAL_LockTypeDef Lock; /* Locking object */
__IO HAL_USART_StateTypeDef State; /* Usart communication state */
__IO HAL_USART_ErrorTypeDef ErrorCode;/* USART Error code */
}USART_HandleTypeDef;
```

1) The multi-instance feature implies that all the APIs used in the application are re-entrant and avoid using global variables because subroutines can fail to be re-entrant if they rely on a global variable to remain unchanged but that variable is modified when the subroutine is recursively invoked. For this reason, the following rules are respected:

- Re-entrant code does not hold any static (or global) non-constant data: re-entrant functions can work with global data. For example, a re-entrant interrupt service routine can grab a piece of hardware status to work with (e.g. serial port read buffer) which is not only global, but volatile. Still, typical use of static variables and global data is not advised, in the sense that only atomic read-modify-write instructions should be used in these variables. It should not be possible for an interrupt or signal to occur during the execution of such an instruction.
- Reentrant code does not modify its own code.

2) When a peripheral can manage several processes simultaneously using the DMA (full duplex case), the DMA interface handle for each process is added in the PPP_HandleTypeDef.

3) For the shared and system peripherals, no handle or instance object is used. The peripherals concerned by this exception are the following:

- GPIO
- SYSTICK
- NVIC
- PWR
- RCC
- FLASH.

2.2.2 Initialization and configuration structure

These structures are defined in the generic driver header file when it is common to all part numbers. When they can change from one part number to another, the structures are defined in the extension header file for each part number.

```
typedef struct
{
uint32_t BaudRate; /*!< This member configures the UART communication baudrate.*/
uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received
in a frame.*/
uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
uint32_t Parity; /*!< Specifies the parity mode. */
uint32_t Mode; /*!< Specifies wether the Receive or Transmit mode is enabled or
disabled.*/
uint32_t HwFlowCtl; /*!< Specifies wether the hardware flow control mode is enabled
or disabled.*/
uint32_t OverSampling; /*!< Specifies wether the Over sampling 8 is enabled or
disabled,
to achieve higher speed (up to fPCLK/8).*/
}UART_InitTypeDef;
```

The config structure is used to initialize the sub-modules or sub-instances. See below example:

```
HAL_ADC_ConfigChannel (ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef*
sConfig)
```

2.2.3 Specific process structures

The specific process structures are used for specific process (common APIs). They are defined in the generic driver header file.

Example:

```
HAL_PPP_Process (PPP_HandleTypeDef* hadc,PPP_ProcessConfig* sConfig)
```

2.3 API classification

The HAL APIs are classified into three categories:

- **Generic APIs:** common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL driver files of all STM32 microcontrollers.

```
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc); HAL_StatusTypeDef
HAL_ADC_DeInit(ADC_HandleTypeDef *hadc); HAL_StatusTypeDef
HAL_ADC_Start(ADC_HandleTypeDef* hadc); HAL_StatusTypeDef
HAL_ADC_Stop(ADC_HandleTypeDef* hadc); HAL_StatusTypeDef
HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc); HAL_StatusTypeDef
HAL_ADC_Stop_IT(ADC_HandleTypeDef* hadc); void HAL_ADC_IRQHandler(ADC_HandleTypeDef*
hadc);
```

- **Extension APIs:** This set of API is divided into two sub-categories:
  - **Family specific APIs:** APIs applying to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

```
HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef* hadc, uint32_t
SingleDiff);
uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t
SingleDiff);
```

  - **Device part number specific APIs:** These APIs are implemented in the extension file and delimited by specific define statements relative to a given part number.

```
#if defined(STM32L475xx) || defined(STM32L476xx)|| defined(STM32L486xx)
void HAL_PWREx_EnableVddUSB(void);
void HAL_PWREx_DisableVddUSB(void);
#endif /* STM32L475xx || STM32L476xx || STM32L486xx */
```

The data structure related to the specific APIs is delimited by the device part number define statement. It is located in the corresponding extension header C file.

The following table summarizes the location of the different categories of HAL APIs in the driver files.

Table 4: API classification

	Generic file	Extension file
Common APIs	X	X
Family specific APIs		X
Device specific APIs		X

Family specific APIs are only related to a given family. This means that if a specific API is implemented in another family, and the arguments of this latter family are different, additional structures and arguments might need to be added.

The IRQ handlers are used for common and family specific processes.

2.4 Devices supported by HAL drivers

Table 5: List of STM32L4 Series devices supported by HAL drivers

IP/ Module	STM32L431xx	STM32L432xx	STM32L442xx	STM32L433xx	STM32L443xx	STM32L451xx	STM32L452xx	STM32L462xx	STM32L471xx	STM32L475xx	STM32L476xx	STM32L485xx	STM32L486xx	STM32L496xx	STM32L4A6xx
stm32l4xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ adc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ can.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ comp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ cortex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ crc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ cryp.c	No	No	Yes	No	Yes	No	No	Yes	No	No	No	Yes	Yes	No	Yes
stm32l4xx_hal_ cryp_ex.c	No	No	Yes	No	Yes	No	No	Yes	No	No	No	Yes	Yes	No	Yes
stm32l4xx_hal_ dac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ dac_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ dcmi.c	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes
stm32l4xx_hal_ dfsdm.c	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ dfsdm_ex.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_ dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ dma_ex.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_ dma2d.c	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes
stm32l4xx_hal_ dsi.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_ firewall.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ flash.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ flash_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/ Module	STM32L431xx	STM32L432xx	STM32L442xx	STM32L433xx	STM32L443xx	STM32L451xx	STM32L452xx	STM32L462xx	STM32L471xx	STM32L475xx	STM32L476xx	STM32L485xx	STM32L486xx	STM32L496xx	STM32L4A6xx
stm32l4xx_hal_ flash_ramfunc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ gfxmmu.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_ gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ hash.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes
stm32l4xx_hal_ hash_ex.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes
stm32l4xx_hal_ hcd.c	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ i2c_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ irda.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ lcd.c	No	No	No	Yes	Yes	No	No	No	No	No	Yes	No	Yes	Yes	Yes
stm32l4xx_hal_ lptim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ ltdc.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_ ltdc_ex.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_ msp_template.c	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stm32l4xx_hal_ nand.c	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ nor.c	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_op amp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_op amp_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ ospi.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_ pcd.c	No	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ pcd_ex.c	No	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/ Module	STM32L431xx	STM32L432xx	STM32L442xx	STM32L433xx	STM32L443xx	STM32L451xx	STM32L452xx	STM32L462xx	STM32L471xx	STM32L475xx	STM32L476xx	STM32L485xx	STM32L486xx	STM32L496xx	STM32L4A6xx
stm32l4xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_qspi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_rng.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_sai.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_sai_ex.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_sd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_sd_ex.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_smartcard_ex.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_smbus.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_spi_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_sram.c	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_swpmi.c	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_tim_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_tsc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_uart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/ Module	STM32L431xx	STM32L432xx	STM32L442xx	STM32L433xx	STM32L443xx	STM32L451xx	STM32L452xx	STM32L462xx	STM32L471xx	STM32L475xx	STM32L476xx	STM32L485xx	STM32L486xx	STM32L496xx	STM32L4A6xx
stm32l4xx_hal_ usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ usart_ex.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
stm32l4xx_hal_ wwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 6: List of STM32L4+ Series devices supported by HAL drivers

IP/Module	STM32L4R5xx	STM32L4R7xx	STM32L4R9xx	STM32L4S5xx	STM32L4S7xx	STM32L4S9xx
stm32l4xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_adc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_adc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_can.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_comp.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_cortex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_crc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_crc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_cryp.c	No	No	No	Yes	Yes	Yes
stm32l4xx_hal_cryp_ex.c	No	No	No	Yes	Yes	Yes
stm32l4xx_hal_dac.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_dac_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_dcmi.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_dfsdm.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_dfsdm_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_dma_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_dma2d.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_dsi.c	No	No	Yes	No	No	Yes
stm32l4xx_hal_firewall.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_flash.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_flash_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_flash_ramfunc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_gfxmmu.c	No	Yes	Yes	No	Yes	Yes
stm32l4xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32L4R5xx	STM32L4R7xx	STM32L4R9xx	STM32L4S5xx	STM32L4S7xx	STM32L4S9xx
stm32l4xx_hal_hash.c	No	No	No	Yes	Yes	Yes
stm32l4xx_hal_hash_ex.c	No	No	No	Yes	Yes	Yes
stm32l4xx_hal_hcd.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_i2c_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_lcd.c	No	No	No	No	No	No
stm32l4xx_hal_lptim.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ltdc.c	No	Yes	Yes	No	Yes	Yes
stm32l4xx_hal_ltdc_ex.c	No	Yes	Yes	No	Yes	Yes
stm32l4xx_hal_msp_template.c	NA	NA	NA	NA	NA	NA
stm32l4xx_hal_nand.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_nor.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_opamp.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_opamp_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_ospi.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_pcd.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_pcd_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_qspi.c	No	No	No	No	No	No
stm32l4xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_rng.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_sai.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_sai_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_sd.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_sd_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_smartcard_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_smbus.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_spi_ex.c	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32L4R5xx	STM32L4R7xx	STM32L4R9xx	STM32L4S5xx	STM32L4S7xx	STM32L4S9xx
stm32l4xx_hal_sram.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_swpmi.c	No	No	No	No	No	No
stm32l4xx_hal_tim.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_tim_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_tsc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_uart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_usart.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_usart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l4xx_hal_wwdg.c	Yes	Yes	Yes	Yes	Yes	Yes

2.5 HAL driver rules

2.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 7: HAL API naming rules

	Generic	Family specific	Device specific
File names	*stm32l4xx_hal_ppp (c/h)*	*stm32l4xx_hal_ppp_ex (c/h)*	*stm32l4xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function* *HAL_PPP_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameTypeDef*	*NA*	*NA*

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.

- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32L4 series and STM32L4+ series reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (e.g. ADC_TypeDef) in stm32l4xxx.h header file.stm32l4xxx.h corresponds to stm32l471xx.h,stm32l475xx.h, stm32l476xx, stm32l485xx and stm32l486xx.h.
- Peripheral function names are prefixed by HAL_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (e.g. HAL_UART_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP_InitTypeDef (e.g. ADC_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP_xxxxConfTypeDef (e.g. ADC_ChannelConfTypeDef).
- Peripheral handle structures are named PPP_HandleTypedef (e.g DMA_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP_InitTypeDef are named HAL_PPP_Init (e.g. HAL_TIM_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named HAL_PPP_DeInit (e.g. HAL_TIM_DeInit()).
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL_PPP_Function_DMA ().*
- The **Feature** prefix should refer to the new feature.
  Example: *HAL_ADCEx_InjectedStart()*() refers to the injection mode

2.5.2 HAL general naming rules

- For the shared and system peripherals, no handle or instance object is used. This rule applies to the following peripherals:
  - GPIO
  - SYSTICK
  - NVIC
  - RCC
  - FLASH.

Example: The *HAL_GPIO_Init()* requires only the GPIO address and its configuration parameters.

```
HAL_StatusTypeDef HAL_GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef *Init)
{
/*GPIO Initialization body */
}
```

- The macros that handle interrupts and specific clock configurations are defined in each peripheral/module driver. These macros are exported in the peripheral driver header files so that they can be used by the extension file. The list of these macros is defined below: This list is not exhaustive and other macros related to peripheral features can be added, so that they can be used in the user application.

Table 8: Macros handling interrupts and specific clock configurations

Macros	Description
__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)	Enables a specific peripheral interrupt
__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)	Disables a specific peripheral interrupt
__HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)	Gets a specific peripheral interrupt status
__HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)	Clears a specific peripheral interrupt status
__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)	Gets a specific peripheral flag status
__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)	Clears a specific peripheral flag status
__HAL_PPP_ENABLE(__HANDLE__)	Enables a peripheral
__HAL_PPP_DISABLE(__HANDLE__)	Disables a peripheral
__HAL_PPP_XXXX (__HANDLE__, __PARAM__)	Specific PPP HAL driver macro
__HAL_PPP_GET_ IT_SOURCE (__HANDLE__, __ INTERRUPT __)	Checks the source of specified interrupt

- NVIC and SYSTICK are two Arm Cortex core features. The APIs related to these features are located in the stm32l4xx_hal_cortex.c file.
- When a status bit or a flag is read from registers, it is composed of shifted values depending on the number of read values and of their size. In this case, the returned status width is 32 bits. Example: STATUS = XX | (YY << 16) or STATUS = XX | (YY << 8) | (YY << 16) | (YY << 24)".
- The PPP handles are valid before using the HAL_PPP_Init() API. The init function performs a check before modifying the handle fields.

```
HAL_PPP_Init(PPP_HandleTypeDef) if(hppp == NULL) { return HAL_ERROR; }
```

- The macros defined below are used:
  – Conditional macro:

```
#define ABS(x) (((x) > 0) ? (x): -(x))
```

  – Pseudo-code macro (multiple instructions macro):

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \µ
 do{ \ (__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
 (__DMA_HANDLE_).Parent = (__HANDLE__); \
 } while(0)
```

2.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32l4xx_it.c
- User callback functions.

The user callback functions are defined as empty functions with “weak” attribute. They have to be defined in the user code.

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: HAL_PPP_MspInit() and HAL_PPP_MspDeInit
- Process complete callbacks: HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 9: Callback functions

Callback functions	Example
HAL_PPP_MspInit() / _DeInit()	Ex: HAL_USART_MspInit() Called from HAL_PPP_Init() API function to perform peripheral system level initialization (GPIOs, clock, DMA, interrupt)
HAL_PPP_ProcessCpltCallback	Ex: HAL_USART_TxCpltCallback Called by peripheral or DMA interrupt handler when the process completes
HAL_PPP_ErrorCallback	Ex: HAL_USART_ErrorCallback Called by peripheral or DMA interrupt handler when an error occurs

2.6 HAL generic APIs

The generic APIs provide common generic functions applying to all STM32 devices. They are composed of four APIs groups:

- **Initialization and de-initialization functions:**HAL_PPP_Init(), HAL_PPP_DeInit()
- **IO operation functions**: HAL_PPP_Read(), HAL_PPP_Write(),HAL_PPP_Transmit(), HAL_PPP_Receive()
- **Control functions**: HAL_PPP_Set (), HAL_PPP_Get ().
- **State and Errors functions**: HAL_PPP_GetState (), HAL_PPP_GetError ().

For some peripheral/module drivers, these groups are modified depending on the peripheral/module implementation.

Example: in the timer driver, the API grouping is based on timer features (PWM, OC, IC...).

The initialization and de-initialization functions allow initializing a peripheral and configuring the low-level resources, mainly clocks, GPIO, alternate functions (AF) and possibly DMA and interrupts. The *HAL_DeInit()*function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.

The IO operation functions perform a row access to the peripheral payload data in write and read modes.

The control functions are used to change dynamically the peripheral configuration and set another operating mode.

The peripheral state and errors functions allow retrieving in runtime the peripheral and data flow states, and identifying the type of errors that occurred. The example below is based on the ADC peripheral. The list of generic APIs is not exhaustive. It is only given as an example.

Table 10: HAL generic APIs

Function group	Common API name	Description
Initialization group	*HAL_ADC_Init()*	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
IO operation group	*HAL_ADC_Start ()*	This function starts ADC conversions when the polling method is used
	HAL_ADC_Stop ()	This function stops ADC conversions when the polling method is used
	HAL_ADC_PollForConversion()	This function allows waiting for the end of conversions when the polling method is used. In this case, a timout value is specified by the user according to the application.
	HAL_ADC_Start_IT()	This function starts ADC conversions when the interrupt method is used
	HAL_ADC_Stop_IT()	This function stops ADC conversions when the interrupt method is used
	HAL_ADC_IRQHandler()	This function handles ADC interrupt requests
	HAL_ADC_ConvCpltCallback()	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	HAL_ADC_ErrorCallback()	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	*HAL_ADC_ConfigChannel()*	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	HAL_ADC_AnalogWDGConfig	This function configures the analog watchdog for the selected ADC
State and Errors group	*HAL_ADC_GetState()*	This function allows getting in runtime the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction allows getting in runtime the error that occurred during IT routine

2.7 HAL extension APIs

2.7.1 HAL extension model overview

The extension APIs provide specific functions or overwrite modified APIs for a specific family (series) or specific part number within the same family.

The extension model consists of an additional file, stm32l4xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32l4xx_hal_ppp_ex.h) for a given part number.

Below an example based on the ADC peripheral:

Table 11: HAL extension APIs

Function Group	Common API Name
HAL_ADCEx_CalibrationStart()	This function is used to start the automatic ADC calibration

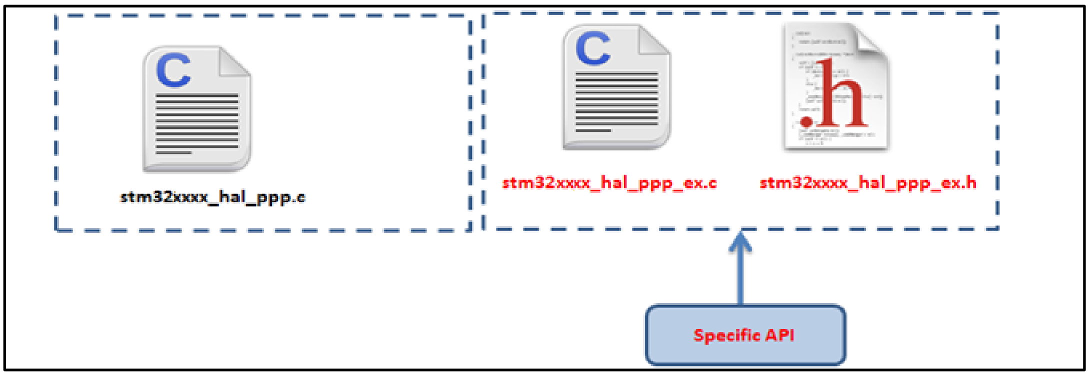
2.7.2 HAL extension model cases

The specific IP features can be handled by the HAL drivers in five different ways. They are described below.

Case 1: Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the stm32l4xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

Figure 2: Adding device-specific functions



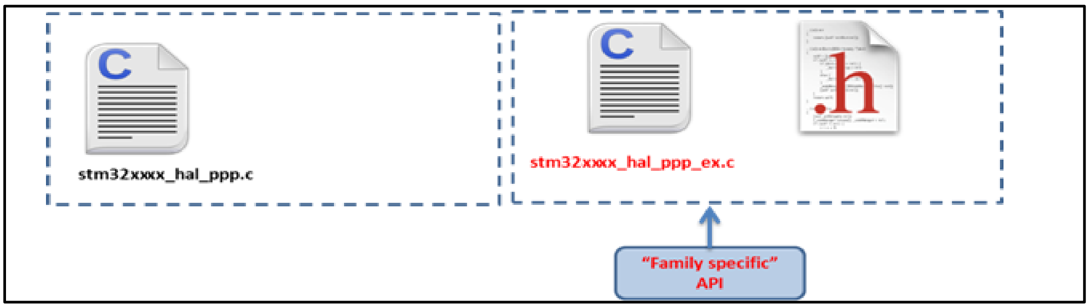
Example: stm32l4xx_hal_adc_ex.c/h

```
#if defined(STM32L475xx) || defined(STM32L476xx) || defined(STM32L486xx)
void HAL_PWREx_EnableVddUSB(void);
void HAL_PWREx_DisableVddUSB(void);
#endif /* STM32L475xx || STM32L476xx || STM32L486xx */
```

Case 2: Adding a family-specific function

In this case, the API is added in the extension driver C file and named HAL_PPPEx_Function ().

Figure 3: Adding family-specific functions

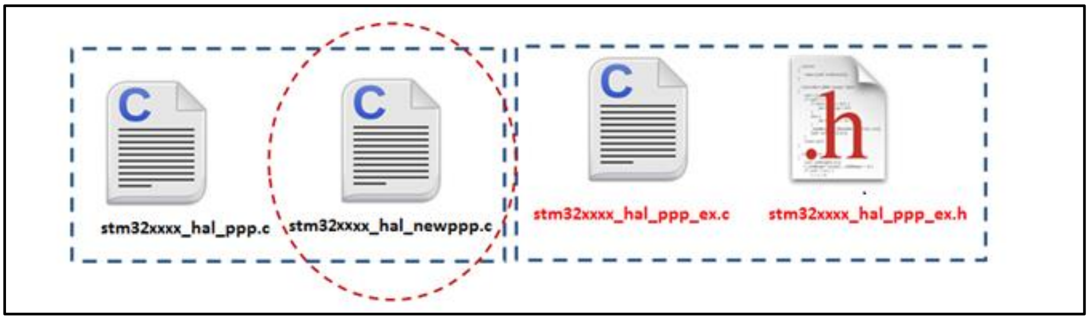


Case 3: Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module (newPPP) are added in a new stm32l4xx_hal_newppp.c. However the inclusion of this file is selected in the stm32lxx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

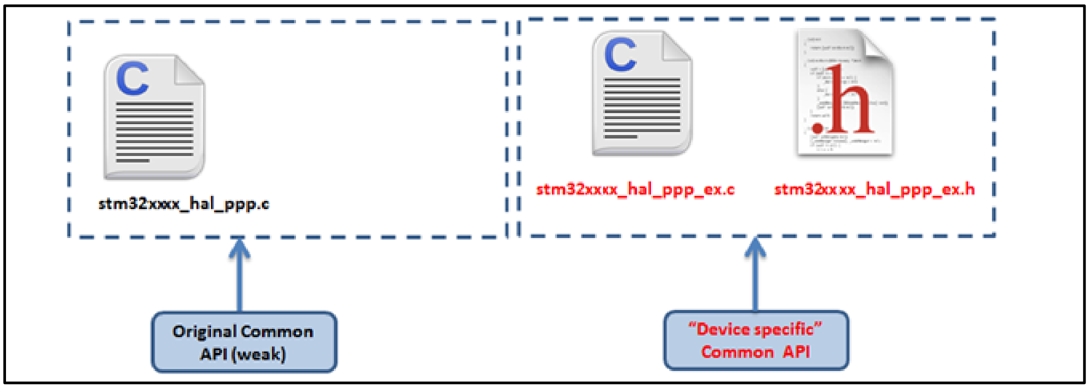


Example: stm32l4xx_hal_lcd.c/h

Case 4: Updating existing common APIs

In this case, the routines are defined with the same names in the stm32l4xx_hal_ppp_ex.c extension file, while the generic API is defined as *weak*, so that the compiler will overwrite the original routine by the new defined function.

Figure 5: Updating existing APIs



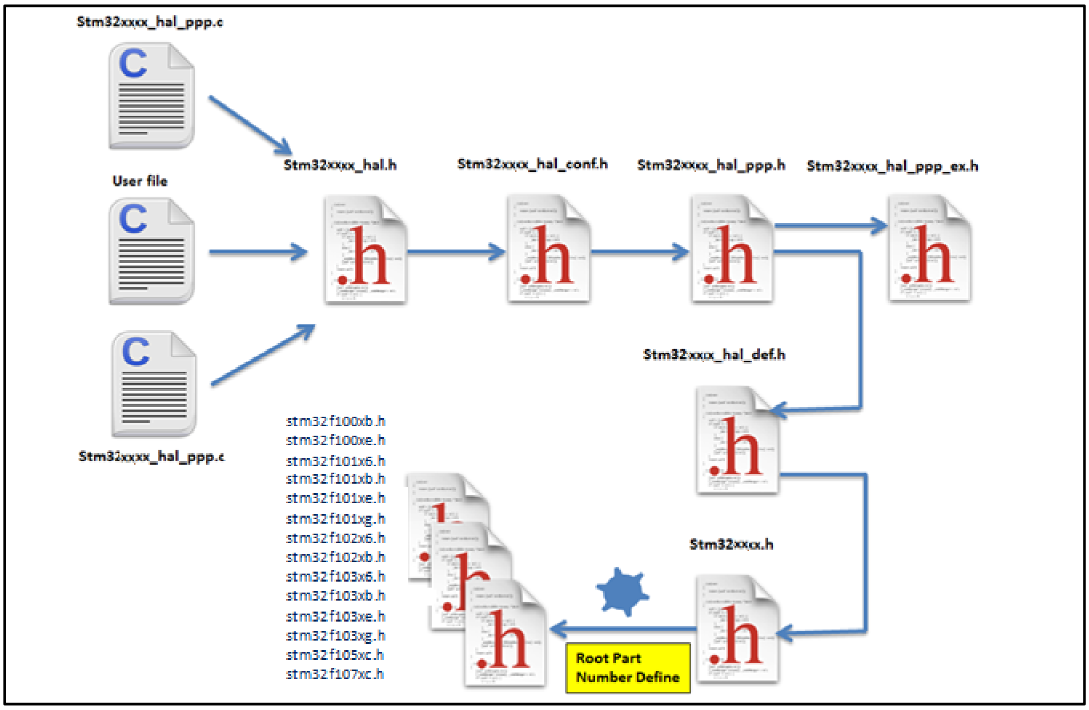
Case 5: Updating existing data structures

The data structure for a specific device part number (e.g. PPP_InitTypeDef) can have different fields. In this case, the data structure is defined in the extension header file and delimited by the specific part number define statement.

2.8 File inclusion model

The header of the common HAL driver file (stm32l4xx_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 6: File inclusion model



A PPP driver is a standalone module which is used in a project. The user must enable the corresponding USE_HAL_PPP_MODULE define statement in the configuration file.

```
/*********************************************************************
* @file stm32l4xx_hal_conf.h
* @author MCD Application Team
* @version VX.Y.Z * @date dd-mm-yyyy
* @brief This file contains the modules to be used
**********************************************************************
(…)
#define HAL_USART_MODULE_ENABLED
#define HAL_IRDA_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
(…)
```

2.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm32l4xx_hal_def.h.*The main common define enumeration is *HAL_StatusTypeDef*.

- **HAL Status** The HAL status is used by almost all HAL APIs, except for boolean functions and IRQ handler. It returns the status of the current API operations. It has four possible values as described below:

```
Typedef enum
{ HAL_OK = 0x00, HAL_ERROR = 0x01, HAL_BUSY = 0x02, HAL_TIMEOUT = 0x03
} HAL_StatusTypeDef;
```

- **HAL Locked** The HAL lock is used by all HAL APIs to prevent accessing by accident shared resources.

```
typedef enum
{ HAL_UNLOCKED = 0x00, /*!<Resources unlocked */
HAL_LOCKED = 0x01 /*!< Resources locked */
} HAL_LockTypeDef;
```

  In addition to common resources, the stm32l4xx_hal_def.h file calls the stm32l4xx.h file in CMSIS library to get the data structures and the address mapping for all peripherals:

  – Declarations of peripheral registers and bits definition.
  – Macros to access peripheral registers hardware (Write register, Read register…etc.).
- **Common macros**
  – Macro defining HAL_MAX_DELAY

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

  – Macro linking a PPP peripheral to a DMA structure pointer:

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
(__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
(__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

2.10 HAL configuration

The configuration file, stm32l4xx_hal_conf.h, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

To configure these parameters, the user should enable, disable or modify some options by uncommenting, commenting or modifying the values of the related define statements as described in the table below:

Table 12: Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	8 000 000 Hz
HSE_STARTUP_TIMEOUT	Timeout for HSE start-up, expressed in ms	100

Configuration item	Description	Default Value
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 Hz
MSI_VALUE	Defines the default value of the Multiplespeed internal oscillator (MSI) expressed in Hz.	4 000 000 Hz
LSI_VALUE	Defines the default value of the Low-speed internal oscillator (LSI) expressed in Hz.	32000 Hz
LSE_VALUE	Defines the value of the external oscillator (LSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 Hz
LSE_STARTUP_TIMEOUT	Timeout for LSE start-up, expressed in ms	5000
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	FALSE
INSTRUCTION_CACHE_ENABLE	Enables I-cache feature	TRUE
DATA_CACHE_ENABLE	Enables D-cache feature	TRUE

The stm32l4xx_hal_conf_template.h file is located in the HAL drivers *Inc* folder. It should be copied to the user folder, renamed and modified as described above.

By default, the values defined in the stm32l4xx_hal_conf_template.h file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

2.11 HAL system peripheral handling

This chapter gives an overview of how the system peripherals are handled by the HAL drivers. The full API list is provided within each peripheral driver description section.

2.11.1 Clock

Two main functions can be used to configure the system clock:

- HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct). This function configures/enables multiple clock sources (HSE, HSI, MSI, LSE, LSI, PLL).
- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency). This function
  - selects the system clock source
  - configures AHB, APB1 and APB2 clock dividers
  - configures the number of Flash memory wait states
  - updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (RTC, USB…). In this case, the clock configuration is performed by an extended API defined in stm32l4xx_hal_rcc_ex.c: *HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)*.

Additional RCC HAL driver functions are available:

- HAL_RCC_DeInit() Clock de-initialization function that returns clock configuration to reset state
- Get clock functions that allow retreiving various clock configurations (system clock, HCLK, PCLK1, PCLK2, …)
- MCO and CSS configuration functions

A set of macros are defined in stm32l4xx_hal_rcc.h and stm32l4xx_hal_rcc_ex.h. They allow executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

- __HAL_PPP_CLK_ENABLE/__HAL_PPP_CLK_DISABLE to enable/disable the peripheral clock
- __HAL_PPP_FORCE_RESET/__HAL_PPP_RELEASE_RESET to force/release peripheral reset
- __HAL_PPP_CLK_SLEEP_ENABLE/__HAL_PPP_CLK_SLEEP_DISABLE to enable/disable the peripheral clock during low power (Sleep) mode.
- __HAL_PPP_IS_CLK_ENABLED/__HAL_PPP_IS_CLK_DISABLED to query about the enabled/disabled status of the peripheral clock.
- __HAL_PPP_IS_CLK_SLEEP_ENABLED/__HAL_PPP_IS_CLK_SLEEP_DISABLED to query about the enabled/disabled status of the peripheral clock during low power (Sleep) mode.

2.11.2 GPIOs

GPIO HAL APIs are the following:

- HAL_GPIO_Init() / HAL_GPIO_DeInit()
- HAL_GPIO_ReadPin() / HAL_GPIO_WritePin()
- HAL_GPIO_TogglePin ().

In addition to standard GPIO modes (input, output, analog), the pin mode can be configured as EXTI with interrupt or event generation.

When selecting EXTI mode with interrupt generation, the user must call HAL_GPIO_EXTI_IRQHandler() from stm32l4xx_it.c and implement HAL_GPIO_EXTI_Callback()

The table below describes the GPIO_InitTypeDef structure field.

Table 13: Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: • GPIO mode – GPIO_MODE_INPUT: Input floating – GPIO_MODE_OUTPUT_PP: Output push-pull – GPIO_MODE_OUTPUT_OD: Output open drain – GPIO_MODE_AF_PP: Alternate Function push-pull – GPIO_MODE_AF_OD: Alternate Function open drain – GPIO_MODE_ANALOG: Analog mode – GPIO_MODE_ANALOG_ADC_CONTROL: ADC analog mode • External Interrupt mode – GPIO_MODE_IT_RISING: Rising edge trigger detection – GPIO_MODE_IT_FALLING: Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING: Rising/Falling edge trigger detection • External Event mode – GPIO_MODE_EVT_RISING: Rising edge trigger detection – GPIO_MODE_EVT_FALLING: Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_FREQ_LOW GPIO_SPEED_FREQ_MEDIUM GPIO_SPEED_FREQ_HIGH GPIO_SPEED_FREQ_VERY_HIGH

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external LEDs:

```
GPIO_InitStruct.Pin = GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

- Configuring PA0 as external interrupt with falling edge sensitivity:

```
GPIO_InitStructure.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Pin = GPIO_PIN_0;
HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
```

2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, stm32l4xx_hal_cortex.c, provides APIs to handle NVIC and Systick. The supported APIs include:

- HAL_NVIC_SetPriority()/ HAL_NVIC_SetPriorityGrouping()
- HAL_NVIC_GetPriority() / HAL_NVIC_GetPriorityGrouping()
- HAL_NVIC_EnableIRQ()/HAL_NVIC_DisableIRQ()
- HAL_NVIC_SystemReset()
- HAL_SYSTICK_IRQHandler()
- HAL_NVIC_GetPendingIRQ() / HAL_NVIC_SetPendingIRQ () / HAL_NVIC_ClearPendingIRQ()
- HAL_NVIC_GetActive(IRQn)
- HAL_SYSTICK_Config()
- HAL_SYSTICK_CLKSourceConfig()
- HAL_SYSTICK_Callback()

2.11.4 PWR

The PWR HAL driver handles power management. The features shared between all STM32 Series are listed below:

- PVD configuration, enabling/disabling and interrupt handling
  - HAL_PWR_ConfigPVD()
  - HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
  - HAL_PWR_PVD_IRQHandler()
  - HAL_PWR_PVDCallback()
- Wakeup pin configuration
  - HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()
- Low-power mode entry
  - HAL_PWR_EnterSLEEPMode()
  - HAL_PWR_EnterSTOPMode() (kept for compatibility with other family but identical to HAL_PWREx_EnterSTOP0Mode() or HAL_PWREx_EnterSTOP1Mode() (see hereafter))
  - HAL_PWR_EnterSTANDBYMode()
- STM32L4 series and STM32L4+ series new low-power management features:
  - HAL_PWREx_EnterSTOP0Mode()
  - HAL_PWREx_EnterSTOP1Mode()
  - HAL_PWREx_EnterSTOP2Mode()
  - HAL_PWREx_EnterSHUTDOWNMode()

2.11.5 EXTI

The EXTI is not considered as a standalone peripheral but rather as a service used by other peripheral. As a result there are no EXTI APIs but each peripheral HAL driver implements the associated EXTI configuration and EXTI function are implemented as macros in its header file.

The first 16 EXTI lines connected to the GPIOs are managed within the GPIO driver. The GPIO_InitTypeDef structure allows configuring an I/O as external interrupt or external event.

The EXTI lines connected internally to the PVD, RTC, USB, and Ethernet are configured within the HAL drivers of these peripheral through the macros given in the table below. The EXTI internal connections depend on the targeted STM32 microcontroller (refer to the product datasheet for more details):

Table 14: Description of EXTI configuration macros

Macros	Description
__HAL_PPP_{SUBLOCK}__EXTI_ENABLE_IT()	Enables a given EXTI line interrupt Example: __HAL_PWR_PVD_EXTI_ENABLE_IT()
__HAL_PPP_{SUBLOCK}__EXTI_DISABLE_IT()	Disables a given EXTI line. Example: __HAL_PWR_PVD_EXTI_DISABLE_IT()
__HAL_ PPP_{SUBLOCK}__EXTI_GET_FLAG()	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PWR_PVD_EXTI_GET_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_CLEAR_FLAG()	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PWR_PVD_EXTI_CLEAR_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_GENERATE_SWIT()	Generates a software interrupt for a given EXTI line. Example: __HAL_PWR_PVD_EXTI_ GENERATE_SWIT ()
__HAL_PPP_SUBBLOCK_EXTI_ENABLE_EVENT()	Enable a given EXTI line event Example: __HAL_RTC_WAKEUP_EXTI_ENABLE_EVENT()
__HAL_PPP_SUBBLOCK_EXTI_DISABLE_EVENT()	Disable a given EXTI line event Example: __HAL_RTC_WAKEUP_EXTI_DISABLE_EVENT()
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_EDGE()	Configure an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()	Enable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_EDGE()	Disable an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()	Disable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_FALLING_ EDGE()	Enable an EXTI Interrupt or Event on Rising/Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_FALLING _EDGE()	Disable an EXTI Interrupt or Event on Rising/Falling edge

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32l4xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback().

2.11.6 DMA

The DMA HAL driver allows enabling and configuring the peripheral to be connected to the DMA Channels (except for internal SRAM/FLASH memory which do not require any initialization). Refer to the product reference manual for details on the DMA request corresponding to each peripheral.

For a given channel, HAL_DMA_Init() API allows programming the required configuration through the following parameters:

- Transfer Direction
- Source and Destination data formats
- Normal or Circular mode
- Channel Priority level
- Source and Destination Increment mode
- Hardware request connected to the peripheral

Two operating modes are available:

- Polling mode I/O operation
  a. Use HAL_DMA_Start() to start DMA transfer when the source and destination addresses and the Length of data to be transferred have been configured.
  b. Use HAL_DMA_PollForTransfer() to poll for the end of current transfer. In this case a fixed timeout can be configured depending on the user application.
- Interrupt mode I/O operation
  a. Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
  b. Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
  c. Use HAL_DMA_Start_IT() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been confgured. In this case the DMA interrupt is configured.
  d. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
  e. When data transfer is complete, HAL_DMA_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (i.e. a member of DMA handle structure).

Additional functions and macros are available to ensure efficient DMA management:

- Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
- Use HAL_DMA_Abort() function to abort the current transfer

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: enables the specified DMA channel.
- __HAL_DMA_DISABLE: disables the specified DMA channel.
- __HAL_DMA_GET_FLAG: gets the DMA channel pending flags.
- __HAL_DMA_CLEAR_FLAG: clears the DMA channel pending flags.
- __HAL_DMA_ENABLE_IT: enables the specified DMA channel interrupts.
- __HAL_DMA_DISABLE_IT: disables the specified DMA channel interrupts.
- __HAL_DMA_GET_IT_SOURCE: checks whether the specified DMA channel interrupt has been enabled or not.

When a peripheral is used in DMA mode, the DMA initialization should be done in the HAL_PPP_MspInit() callback. In addition, the user application should associate the DMA handle to the PPP handle (refer to section "HAL IO operation functions").

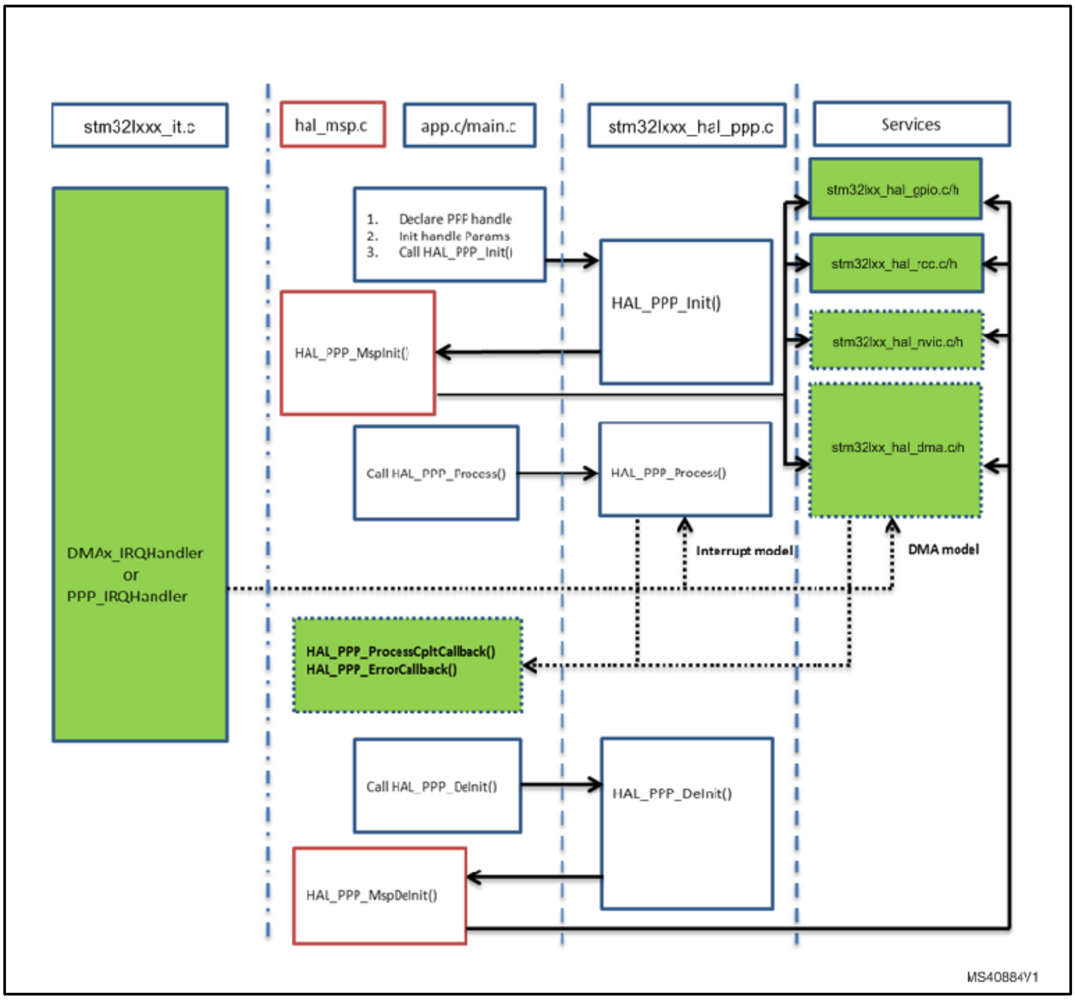
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

2.12 How to use HAL drivers

2.12.1 HAL usage models

The following figure shows the typical use of the HAL driver and the interaction between the application user, the HAL driver and the interrupts.

Figure 7: HAL driver model



The functions implemented in the HAL driver are shown in green, the functions called from interrupt handlers in dotted lines, and the msp functions implemented in the user application in red. Non-dotted lines represent the interactions between the user application functions.

Basically, the HAL driver APIs are called from user files and optionally from interrupt handlers file when the APIs based on the DMA or the PPP peripheral dedicated interrupts are used.

When DMA or PPP peripheral interrupts are used, the PPP process complete callbacks are called to inform the user about the process completion in real-time event mode (interrupts). Note that the same process completion callbacks are used for DMA in interrupt mode.

2.12.2 HAL initialization

2.12.2.1 HAL global initialization

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file stm32l4xx_hal.c.

- HAL_Init(): this function must be called at application startup to
  - initialize data/instruction cache and pre-fetch queue
  - set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
  - call HAL_MspInit() user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). HAL_MspInit() is defined as “weak” empty function in the HAL drivers.
- HAL_DeInit()
  - resets all peripherals
  - calls function HAL_MspDeInit() which a is user callback function to do system level De-Initalizations.
- HAL_GetTick(): this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.
  Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be blocked.

2.12.2.2 System clock initialization

The clock configuration is done at the beginning of the user code. However the user can change the configuration of the clock in his own code. Please find below the typical Clock configuration sequence to reach the maximum 80 MHz clock frequency based on the HSE clock:

```
 void SystemClock_Config(void)
{
   RCC_ClkInitTypeDef clkinitstruct = {0};
   RCC_OscInitTypeDef oscinitstruct = {0};
/* Configure PLLs------------------------------------------------------*/
/* PLL configuration: PLLCLK =  (HSE/PLLM * PLLN) / PLLR = (16/1 * 20) / 2 = 80
MHz*/
/* Enable HSE Oscillator and activate PLL with HSE as source */
   oscinitstruct.OscillatorType =  RCC_OSCILLATORTYPE_HSE;
   oscinitstruct.HSEState =   RCC_HSE_ON;
   oscinitstruct.PLL.PLLState = RCC_PLL_ON;
   oscinitstruct.PLL.PLLSource =   RCC_PLLSOURCE_HSE;
   oscinitstruct.PLL.PLLM = 1;
   oscinitstruct.PLL.PLLN = 20;
   oscinitstruct.PLL.PLLR = 2;
   oscinitstruct.PLL.PLLL = 7;
   oscinitstruct.PLL.PLLQ = 4;
   if   (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)
```

```
   {
    /* Initialization Error */
       while(1);
   }
   /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
clocks dividers    */
   clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK I   RCC_CLOCKTYPE_HCLK I
RCC_CLOCKTYPE_PCLK1 I RCC_CLOCKTYPE_PCLK2);
   clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
   clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
   clkinitstruct.APB2CLKDivider =   RCC_HCLK_DIV1;
   clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV1;
   if
      (HAL_RCC_ClockConfig(&clkinitstruct,FLASH_LATENCY_4)!= HAL_OK)
    {
      /*   Initialization Error */
     while(1);
    }
 }
```

2.12.2.3 HAL MSP initialization process

The peripheral initialization is done through *HAL_PPP_Init()* while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling MSP callback function *HAL_PPP_MspInit().*

The MspInit callback performs the low level initialization related to the different additional hardware resources: RCC, GPIO, NVIC and DMA.

All the HAL drivers with handles include two MSP callbacks for initialization and de-initialization:

```
/**
* @brief Initializes the PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspInit(PPP_HandleTypeDef *hppp) {
/* NOTE: This function Should not be modified, when the callback is needed,
the HAL_PPP_MspInit could be implemented in the user file */
}
/**
* @brief DeInitializes PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspDeInit(PPP_HandleTypeDef *hppp) {
/* NOTE: This function Should not be modified, when the callback is needed,
the HAL_PPP_MspDeInit could be implemented in the user file */
}
```

The MSP callbacks are declared empty as weak functions in each peripheral driver. The user can use them to set the low level initialization code or omit them and use his own initialization routine.

The HAL MSP callback is implemented inside the *stm32l4xx_hal_msp.c* file in the user folders. An *stm32l4xx_hal_msp.c* file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low level initialization code.

stm32l4xx_hal_msp.c file contains the following functions:

Table 15: MSP functions

Routine	Description
void HAL_MspInit()	Global MSP initialization routine
void HAL_MspDeInit()	Global MSP de-initialization routine

Routine	Description
void HAL_PPP_MspInit()	PPP MSP initialization routine
void HAL_PPP_MspDeInit()	PPP MSP de-initialization routine

By default, if no peripheral needs to be de-initialized during the program execution, the whole MSP initialization is done in *Hal_MspInit()* and MSP De-Initialization in the *Hal_MspDeInit().* In this case the *HAL_PPP_MspInit()* and *HAL_PPP_MspDeInit()* are not implemented.

When one or more peripherals needs to be de-initialized in run time and the low level resources of a given peripheral need to be released and used by another peripheral, HAL_PPP_MspDeInit() and *HAL_PPP_MspInit()* are implemented for the concerned peripheral and other peripherals initialization and de-Initialization are kept in the global *HAL_MspInit()* and the *HAL_MspDeInit().*

If there is nothing to be initialized by the global *HAL_MspInit()* and *HAL_MspDeInit()*, the two routines can simply be omitted.

2.12.3 HAL IO operation process

The HAL functions with internal data processing like transmit, receive, write and read are generally provided with three data processing modes as follows:

- Polling mode
- Interrupt mode
- DMA mode

2.12.3.1 Polling mode

In Polling mode, the HAL functions return the process status when the data processing in blocking mode is complete. The operation is considered complete when the function returns the HAL_OK status, otherwise an error status is returned. The user can get more information through the *HAL_PPP_GetState()* function. The data processing is handled internally in a loop. A timeout (expressed in ms) is used to prevent process hanging.

The example below shows the typical Polling mode processing sequence:

```
HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t pData,
int16_tSize,uint32_tTimeout)
{
if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(…) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(…)
return HELIAC; }
```

2.12.3.2 Interrupt mode

In interrupt mode, the HAL function returns the process status after starting the data processing and enabling the appropriate interruption. The end of the operation is indicated by a callback declared as a weak function. It can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function.

In interrupt mode, four functions are declared in the driver:

- *HAL_PPP_Process_IT()*: launch the process
- *HAL_PPP_IRQHandler()*: the global PPP peripheral interruption
- *__weak HAL_PPP_ProcessCpltCallback ()*: the callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: the callback relative to the process Error.

To use a process in interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and HAL_PPP_IRQHandler in *stm32l4xx_it.c.*

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver. This means that the user can declare it again in the application. The function in the driver is not modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_SendIT(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
}
```

*stm32l4xx_it.c*file:

```
extern UART_HandleTypeDef UartHandle;
void USART1_IRQHandler(void)
{
HAL_UART_IRQHandler(&UartHandle);
}
```

2.12.3.3 DMA mode

In DMA mode, the HAL function returns the process status after starting the data processing through the DMA and after enabling the appropriate DMA interruption. The end of the operation is indicated by a callback declared as a weak function and can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function. For the DMA mode, three functions are declared in the driver:

- *HAL_PPP_Process_DMA()*: launch the process
- *HAL_PPP_DMA_IRQHandler()*: the DMA interruption used by the PPP peripheral
- *__weak HAL_PPP_ProcessCpltCallback()*: the callback relative to the process completion.
- *__weak HAL_PPP_ErrorCpltCallback()*: the callback relative to the process Error.

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *stm32l4xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
PPP_TypeDef *Instance; /* Register base address */
PPP_InitTypeDef Init; /* PPP communication parameters */
HAL_StateTypeDef State; /* PPP communication state */
(…)
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

```
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = UART1;
HAL_UART_Init(&UartHandle);
(..)
}
void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
static DMA_HandleTypeDef hdma_tx;
static DMA_HandleTypeDef hdma_rx;
(…)
__HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
__HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
(…)
}
```

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver that means, the user can declare it again in the application code. The function in the driver should not be modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Paramaters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX; UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
}
void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
}
```

stm32l4xx_it.c file:

```
extern UART_HandleTypeDef UartHandle;
void DMAx_IRQHandler(void)
{
HAL_DMA_IRQHandler(&UartHandle.DMA_Handle_tx);
}
```

HAL_USART_TxCpltCallback() and *HAL_USART_ErrorCallback()* should be linked in the *HAL_PPP_Process_DMA()* function to the DMA transfer complete callback and the DMA transfer Error callback by using the following statement:

```
HAL_PPP_Process_DMA (PPP_HandleTypeDef *hppp, Params….)
{
(…)
hppp->DMA_Handle->XferCpltCallback = HAL_UART_TxCpltCallback ;
hppp->DMA_Handle->XferErrorCallback = HAL_UART_ErrorCallback ;
(…)
}
```

2.12.4 Timeout and error management

2.12.4.1 Timeout management

The timeout is often used for the APIs that operate in polling mode. It defines the delay during which a blocking process should wait till an error is returned. An example is provided below:

```
HAL_StatusTypeDef HAL_DMA_PollForTransfer(DMA_HandleTypeDef *hdma, uint32_t
CompleteLevel, uint32_t Timeout)
```

The timeout possible value are the following:

Table 16: Timeout values

Timeout value	Description
0	No poll: Immediate process check and exit
1 ... (HAL_MAX_DELAY -1)[(1)]	Timeout in ms
HAL_MAX_DELAY	Infinite poll till process is successful

Notes:

(1)HAL_MAX_DELAY is defined in the stm32l4xx_hal_def.h as 0xFFFFFFFF

However, in some cases, a fixed timeout is used for system peripherals or internal HAL driver processes. In these cases, the timeout has the same meaning and is used in the same way, except when it is defined locally in the drivers and cannot be modified or introduced as an argument in the user application.

Example of fixed timeout:

```
#define LOCAL_PROCESS_TIMEOUT 100
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef)
{
(…)
timeout = HAL_GetTick() + LOCAL_PROCESS_TIMEOUT;
(…)
while(ProcessOngoing)
{
(…)
if(HAL_GetTick() >= timeout)
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return HAL_PPP_STATE_TIMEOUT;
}
}
(…)
}
```

The following example shows how to use the timeout inside the polling functions:

```
HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
(…)
timeout = HAL_GetTick() + Timeout;
(…)
while(ProcessOngoing)
{
(…)
if(Timeout != HAL_MAX_DELAY)
{
if(HAL_GetTick() >= timeout)
```

```
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return hppp->State;
}
}
(…)
}
```

2.12.4.2 Error management

The HAL drivers implement a check on the following items:

- Valid parameters: for some process the used parameters should be valid and already defined, otherwise the system may crash or go into an undefined state. These critical parameters are checked before being used (see example below).

```
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pdata, uint32
Size)
{ if ((pData == NULL ) || (Size == 0))
{ return HAL_ERROR;
}
}
```

- Valid handle: the PPP peripheral handle is the most important argument since it keeps the PPP driver vital parameters. It is always checked in the beginning of the *HAL_PPP_Init()* function.

```
HAL_StatusTypeDef HAL_PPP_Init(PPP_HandleTypeDef* hppp)
{ if (hppp == NULL) //the handle should be already allocated
{ return HAL_ERROR;
}
}
```

- Timeout error: the following statement is used when a timeout error occurs:

```
while (Process ongoing)
{ timeout = HAL_GetTick() + Timeout;
while (data processing is running)
{ if(timeout)
{ return HAL_TIMEOUT;
}
}
```

When an error occurs during a peripheral process, *HAL_PPP_Process ()* returns with a *HAL_ERROR* status. The HAL PPP driver implements the *HAL_PPP_GetError ()* to allow retrieving the origin of the error.

```
HAL_PPP_ErrorTypeDef HAL_PPP_GetError (PPP_HandleTypeDef *hppp);
```

In all peripheral handles, a *HAL_PPP_ErrorTypeDef* is defined and used to store the last error code.

```
typedef struct
{
PPP_TypeDef * Instance; /* PPP registers base address */
PPP_InitTypeDef Init; /* PPP initialization parameters */
HAL_LockTypeDef Lock; /* PPP locking object */
__IO HAL_PPP_StateTypeDef State; /* PPP state */
__IO HAL_PPP_ErrorTypeDef ErrorCode; /* PPP Error code */
(…)
/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

The error state and the peripheral global state are always updated before returning an error:

```
PPP->State = HAL_PPP_READY;  /* Set the peripheral ready */
PP->ErrorCode = HAL_ERRORCODE ; /* Set the error code */
_HAL_UNLOCK(PPP) ; /* Unlock the PPP resources */
return HAL_ERROR;  /*return with HAL error */
```

HAL_PPP_GetError () must be used in interrupt mode in the error callback:

```
void HAL_PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
ErrorCode = HAL_PPP_GetError (hppp); /* retreive error code */
}
```

2.12.4.3 Run-time checking

The HAL implements run-time failure detection by checking the input values of all HAL driver functions. The run-time checking is achieved by using an *assert_param* macro. This macro is used in all the HAL driver functions which have an input parameter. It allows verifying that the input value lies within the parameter allowed values.

To enable the run-time checking, use the *assert_param* macro, and leave the define ***USE_FULL_ASSERT*** uncommented in *stm32l4xx_hal_conf.h* file.

```
void HAL_UART_Init(UART_HandleTypeDef *huart)
{
(..) /* Check the parameters */
assert_param(IS_UART_INSTANCE(huart->Instance));
assert_param(IS_UART_BAUDRATE(huart->Init.BaudRate));
assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
assert_param(IS_UART_STOPBITS(huart->Init.StopBits));
assert_param(IS_UART_PARITY(huart->Init.Parity));
assert_param(IS_UART_MODE(huart->Init.Mode));
assert_param(IS_UART_HARDWARE_FLOW_CONTROL(huart->Init.HwFlowCtl));
(..)

/** @defgroup UART_Word_Length *
@{
*/
#define UART_WORDLENGTH_8B ((uint32_t)0x00000000)
#define UART_WORDLENGTH_9B ((uint32_t)USART_CR1_M)
#define IS_UART_WORD_LENGTH(LENGTH) (((LENGTH) == UART_WORDLENGTH_8B) ||
\ ((LENGTH) == UART_WORDLENGTH_9B))
```

If the expression passed to the assert_param macro is false, the*assert_failed* function is called and returns the name of the source file and the source line number of the call that failed. If the expression is true, no value is returned.

The *assert_param* macro is implemented in stm32l4xx_hal_conf.h:

```
/* Exported macro ------------------------------------------------------------*/
#ifdef USE_FULL_ASSERT
/**
* @brief The assert_param macro is used for function's parameters check.
* @param expr: If expr is false, it calls assert_failed function
* which reports the name of the source file and the source
* line number of the call that failed.
* If expr is true, it returns no value.
* @retval None */
#define assert_param(expr) ((expr)?(void)0:assert_failed((uint8_t *)__FILE__,
__LINE__))
/* Exported functions --------------------------------------*/
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr)((void)0)
#endif /* USE_FULL_ASSERT */
```

The *assert_failed* function is implemented in the main.c file or in any other user C file:

```
#ifdef USE_FULL_ASSERT /**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None */
void assert_failed(uint8_t* file, uint32_t line)
{
/* User can add his own implementation to report the file name and line number,
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
/* Infinite loop */
while (1)
{
}
}
```

Because of the overhead run-time checking introduces, it is recommended to use it during application code development and debugging, and to remove it from the final application to improve code size and speed.

3 Overview of low-layer drivers

The low-layer (LL) drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or those requiring heavy software configuration and/or complex upper-level stack (such as FSMC, USB or SDMMC).

The LL drivers feature:

- A set of functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Functions to perform peripheral de-initialization (peripheral registers restored to their default values)
- A set of inline functions for direct and atomic register access
- Full independence from HAL since LL drivers can be used either in standalone mode (without HAL drivers) or in mixed mode (with HAL drivers)
- Full coverage of the supported peripheral features.

The low-layer drivers provide hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide one-shot operations that must be called following the programming model described in the microcontroller line reference manual. As a result, the LL services do not implement any processing and do not require any additional memory resources to save their states, counter or data pointers: all the operations are performed by changing the associated peripheral registers content.

3.1 Low-layer files

The low-layer drivers are built around header/C files (one per each supported peripheral) plus five header files for some System and Cortex related features.

Table 17: LL driver files

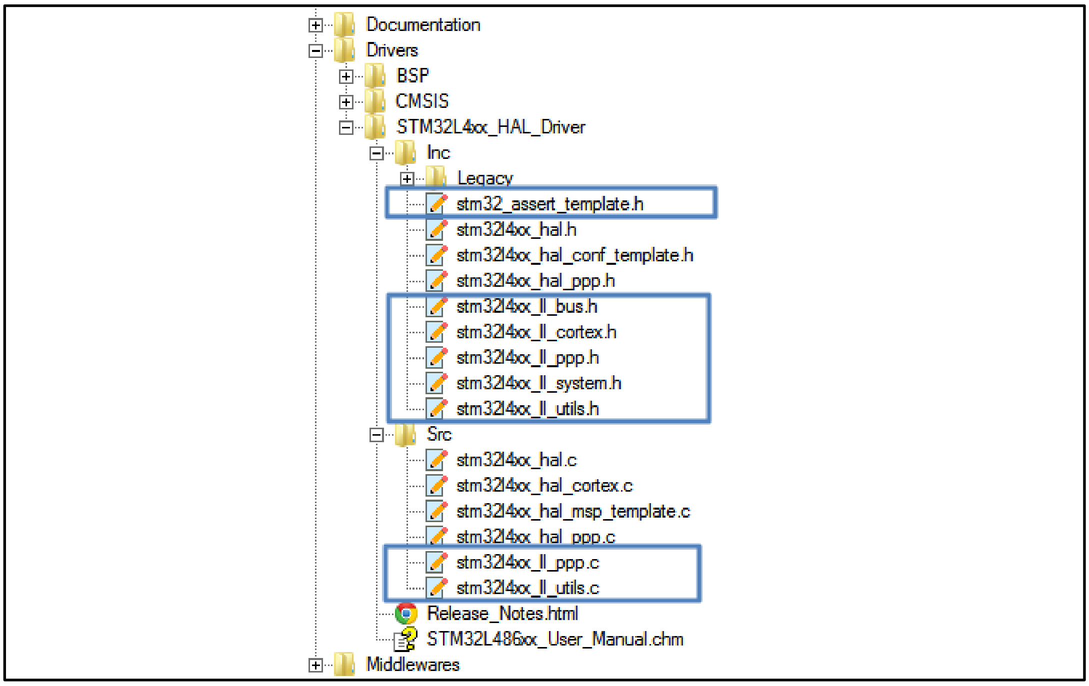
File	Description
stm32l4xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB2_GRP1_EnableClock*
stm32l4xx_ll_ppp.h/.c	stm32l4xx_ll_ppp.c provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DeInit(). All the other APIs are definined within stm32l4xx_ll_ppp.h file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the xx_ll_ppp.h file.
stm32l4xx_ll_cortex.h	Cortex-M related register operation APIs including the Systick, Low power (LL_SYSTICK_xxxxx, LL_LPM_xxxxx "Low Power Mode" ...)
stm32l4xx_ll_utils.h/.c	This file covers the generic APIs: • Read of device unique ID and electronic signature • Timebase and delay management • System clock configuration.
*stm32l4xx_*ll_system.h	System related operations (LL_SYSCFG_xxx, LL_DBGMCU_xxx, LL_FLASH_xxx and LL_VREFBUF_xxx)

File	Description
stm32_assert_template.h	Template file allowing to define the assert_param macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to stm32_assert.h.

There is no configuration file for the LL drivers.

The low-layer files are located in the same HAL driver folder.

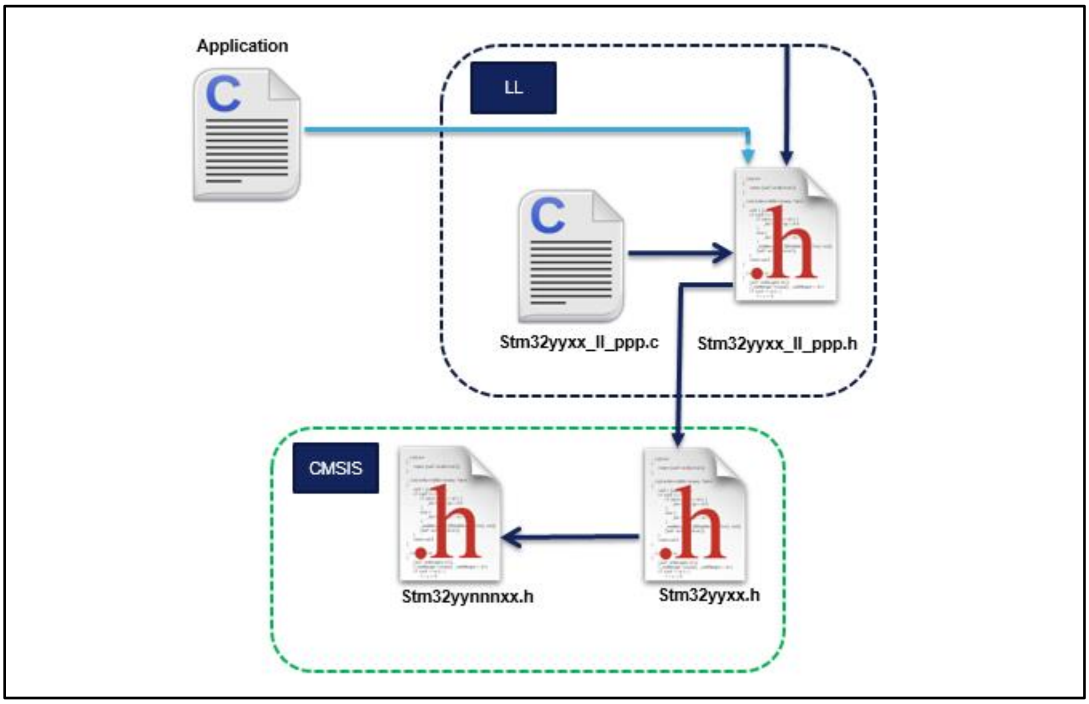
Figure 8: Low-layer driver folders



In general, low-layer drivers include only the STM32 CMSIS device file.

```
#include "stm32yyxx.h"
```

Figure 9: Low-layer driver CMSIS files



Application files have to include only the used low-layer driver header files.

3.2 Overview of low-layer APIs and naming rules

3.2.1 Peripheral initialization functions

The LL drivers offer three set of initialization functions. They are defined in stm32l4xx_ll_ppp.c file:

- Functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Function for peripheral de-initialization (peripheral registers restored to their default values)

The definition of these LL initialization functions and associated resources (structure, literals and prototypes) is conditioned by a compilation switch: *USE_FULL_LL_DRIVER*. To use these functions, this switch must be added in the toolchain compiler preprocessor or to any generic header file which is processed before the LL drivers.

The below table shows the list of the common functions provided for all the supported peripherals:

Table 18: Common peripheral initialization functions

Functions	Return Type	Parameters	Description
LL_PPP_ Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_InitTypeDef* PPP_InitStruct*	Initializes the peripheral main features according to the parameters specified in PPP_InitStruct. Example: LL_USART_Init(USART_TypeDef *USARTx, LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_Struct Init	*void*	• *LL_PPP_InitTypeDef* PPP_InitStruct*	Fills each PPP_InitStruct member with its default value. Example. LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_DeInit	*ErrorStatus*	• *PPP_TypeDef* PPPx*	De-initializes the peripheral registers, that is restore them to their default reset values. Example. LL_USART_DeInit(USART_TypeDef *USARTx)

Additional functions are available for some peripherals (refer to *Table 19: "Optional peripheral initialization functions"*).

Table 19: Optional peripheral initialization functions

Functions	Return Type	Parameters	Examples
LL_PPP{_CATEGORY}_Init	*Error Status*	• *PPP_TypeDef* PPPx* • *LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Initializes peripheral features according to the parameters specified in PPP_InitStruct. Example: LL_ADC_INJ_Init(ADC_TypeDef *ADCx, LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct) LL_RTC_TIME_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef *RTC_TimeStruct) LL_RTC_DATE_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef *RTC_DateStruct) LL_TIM_IC_Init(TIM_TypeDef* TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef* TIM_IC_InitStruct) LL_TIM_ENCODER_Init(TIM_TypeDef* TIMx, LL_TIM_ENCODER_InitTypeDef* TIM_EncoderInitStruct)
LL_PPP{_CATEGORY}_StructInit	*void*	*LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Fills each *PPP{_CATEGORY}_InitStruct* member with its default value. Example: LL_ADC_INJ_StructInit(LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct)
LL_PPP_CommonInit	*Error Status*	• *PPP_TypeDef* PPPx* • *LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Initializes the common features shared between different instances of the same peripheral. Example: LL_ADC_CommonInit(ADC_Common_TypeDef *ADCxy_COMMON, LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)

Functions	**Return Type**	**Parameters**	**Examples**
LL_PPP_CommonStructInit	*void*	*LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Fills each PPP_CommonInitStruct member with its default value Example: LL_ADC_CommonStructInit(LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_ClockInit	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Initializes the peripheral clock configuration in synchronous mode. Example: LL_USART_ClockInit(USART_TypeDef *USARTx, LL_USART_ClockInitTypeDef *USART_ClockInitStruct)
LL_PPP_ClockStructInit	*void*	*LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Fills each *PPP_ClockInitStruct* member with its default value Example: LL_USART_ClockStructInit(LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

3.2.1.1 Run-time checking

Like HAL drivers, LL initialization functions implement run-time failure detection by checking the input values of all LL driver functions. For more details please refer to *Section 2.12.4.3: "Run-time checking"*.

When using the LL drivers in standalone mode (without calling HAL functions), the following actions are required to use run-time checking:

1. Copy stm32_assert_template.h to the application folder and rename it to stm32_assert.h. This file defines the assert_param macro which is used when run-time checking is enabled.
2. Include stm32_assert.h file within the application main header file.
3. Add the USE_FULL_ASSERT compilation switch in the toolchain compiler preprocessor or in any generic header file which is processed before the stm32_assert.h driver.

Run-time checking is not available for LL inline functions.

3.2.2 Peripheral register-level configuration functions

On top of the peripheral initialization functions, the LL drivers offer a set of inline functions for direct atomic register access. Their format is as follows:

```
__STATIC_INLINE return_type LL_PPP_Function (PPPx_TypeDef *PPPx, args)
```

The “Function” naming is defined depending to the action category:

- **Specific Interrupt, DMA request and status flags management**: Set/Get/Clear/Enable/Disable flags on interrupt and status registers

Table 20: Specific Interrupt, DMA request and status flags management

Name	Examples
LL_PPP_{_CATEGORY}_ActionItem_BITNAME *LL_PPP{_CATEGORY}_IsItem_BITNAME_Action*	• LL_RCC_IsActiveFlag_LSIRDY • LL_RCC_IsActiveFlag_FWRST() • LL_ADC_ClearFlag_EOC(ADC1) • LL_DMA_ClearFlag_TCx(DMA_TypeDef* DMAx)

Table 21: Available function formats

Item	Action	Format
Flag	Get	*LL_PPP_IsActiveFlag_BITNAME*
	Clear	*LL_PPP_ClearFlag_BITNAME*
Interrupts	Enable	*LL_PPP_EnableIT_BITNAME*
	Disable	*LL_PPP_DisableIT_BITNAME*
	Get	*LL_PPP_IsEnabledIT_BITNAME*
DMA	Enable	*LL_PPP_EnableDMAReq_BITNAME*
	Disable	*LL_PPP_DisableDMAReq_BITNAME*
	Get	*LL_PPP_IsEnabledDMAReq_BITNAME*

BITNAME refers to the peripheral register bit name as described in the product line reference manual.

- **Peripheral clock activation/deactivation management**: Enable/Disable/Reset a peripheral clock

Table 22: Peripheral clock activation/deactivation management

Name	Examples
LL_BUS_GRPx_ActionClock{Mode}	• *LL_AHB2_GRP1_EnableClock (LL_AHB2_GRP1_PERIPH_GPIOA|LL_AHB2_GRP1_PERIPH_GPIOB)* • *by LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)*

'x' corresponds to the group index and refers to the index of the modified register on a given bus.

- ***Peripheral activation/deactivation management***: Enable/disable a peripheral or activate/deactivate specific peripheral features

Table 23: Peripheral activation/deactivation management

Name	Examples
LL_PPP{_CATEGORY}_Action{Item} *LL_PPP{_CATEGORY}_IsItemAction*	• *LL_ADC_Enable ()* • *LL_ADC_StartCalibration();* • *LL_ADC_IsCalibrationOnGoing;* • *LL_RCC_HSI_Enable ()* • *LL_RCC_HSI_IsReady()*

- ***Peripheral configuration management***: Set/get a peripheral configuration settings

Table 24: Peripheral configuration management

Name	Examples
LL_PPP{_CATEGORY}_Set{ or Get}ConfigItem	*LL_USART_SetBaudRate (USART2, Clock, LL_USART_BAUDRATE_9600)*

- ***Peripheral register management***: Write/read the content of a register/retrun DMA relative register address

Table 25: Peripheral register management

Name
LL_PPP_WriteReg(__INSTANCE__, __REG__, __VALUE__)
LL_PPP_ReadReg(__INSTANCE__, __REG__)
*LL_PPP_DMA_GetRegAddr (PPP_TypeDef *PPPx,{Sub Instance if any ex: Channel} , {uint32_t Propriety})*

The Propriety is a variable used to identify the DMA transfer direction or the data register type.

4 Cohabiting of HAL and LL

The low-ayer APIs are designed to be used in standalone mode or combined with the HAL. They cannot be automatically used with the HAL for the same peripheral instance. If you use the LL APIs for a specific instance, you can still use the HAL APIs for other instances. Be careful that the low-layer APIs might overwrite some registers which content is mirrored in the HAL handles.

4.1 Low-layer driver used in standalone mode

The low-layer APIs can be used without calling the HAL driver services. This is done by simply including stm32l4xx_ll_ppp.h in the application files. The LL APIs for a given peripheral are called by executing the same sequence as the one recommended by the programming model in the corresponding product line reference manual. In this case the HAL drivers associated to the used peripheral can be removed from the workspace. However the STM32CubeL4 framework should be used in the same way as in the HAL drivers case which means that System file, startup file and CMSIS should always be used.

When the BSP drivers are included, the used HAL drivers associated with the BSP functions drivers should be included in the workspace, even if they are not used by the application layer.

4.2 Mixed use of low-layer APIs and HAL drivers

In this case the low-layer APIs are used in conjunction with the HAL drivers to achieve direct and register level based operations.

Mixed use is allowed, however some consideration should be taken into account:

- It is recommended to avoid using simultaneously the HAL APIs and the combination of low-layer APIs for a given peripheral instance. If this is the case, one or more private fields in the HAL PPP handle structure should be updated accordingly.
- For operations and processes that do not alter the handle fields including the initialization structure, the HAL driver APIs and the low-layer services can be used together for the same peripheral instance.
- The low-layer drivers can be used without any restriction with all the HAL drivers that are not based on handle objects (RCC, common HAL, flash and GPIO).

Several examples showing how to use HAL and LL in the same application are provided within stm32l4 firmware package (refer to Examples_MIX projects).

1. When the HAL Init/DeInit APIs are not used and are replaced by the low-layer macros, the InitMsp() functions are not called and the MSP initialization should be done in the user application.
2. When process APIs are not used and the corresponding function is performed through the low-layer APIs, the callbacks are not called and post processing or error management should be done by the user application.
3. When the LL APIs is used for process operations, the IRQ handler HAL APIs cannot be called and the IRQ should be implemented by the user application. Each LL driver implements the macros needed to read and clear the associated interrupt flags.

5 HAL System Driver

5.1 HAL Firmware driver API description

5.1.1 How to use this driver

The common HAL driver contains a set of generic and common APIs that can be used by the PPP peripheral drivers and the user to start using the HAL.

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

5.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the Flash interface the NVIC allocation and initial time base clock configuration.
- De-initialize common part of the HAL.
- Configure the time base source to have 1ms time base with a dedicated Tick interrupt priority.
  - SysTick timer is used by default as source of time base, but user can eventually implement his proper time base source (a general purpose timer for example or other time source), keeping in mind that Time base duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and handled in milliseconds basis.
  - Time base configuration function (HAL_InitTick ()) is called automatically at the beginning of the program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().
  - Source of time base is configured to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, the Tick interrupt line must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked.
  - functions affecting time base configurations are declared as __weak to make override possible in case of other implementations in user file.

This section contains the following APIs:

- ***HAL_Init()***
- ***HAL_DeInit()***
- ***HAL_MspInit()***
- ***HAL_MspDeInit()***
- ***HAL_InitTick()***

5.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier
- Get the device revision identifier

This section contains the following APIs:

- *HAL_IncTick()*
- *HAL_GetTick()*
- *HAL_Delay()*
- *HAL_SuspendTick()*
- *HAL_ResumeTick()*
- *HAL_GetHalVersion()*
- *HAL_GetREVID()*
- *HAL_GetDEVID()*
- *HAL_GetUIDw0()*
- *HAL_GetUIDw1()*
- *HAL_GetUIDw2()*

5.1.4 HAL Debug functions

This section provides functions allowing to:

- Enable/Disable Debug module during SLEEP mode
- Enable/Disable Debug module during STOP0/STOP1/STOP2 modes
- Enable/Disable Debug module during STANDBY mode

This section contains the following APIs:

- *HAL_DBGMCU_EnableDBGSleepMode()*
- *HAL_DBGMCU_DisableDBGSleepMode()*
- *HAL_DBGMCU_EnableDBGStopMode()*
- *HAL_DBGMCU_DisableDBGStopMode()*
- *HAL_DBGMCU_EnableDBGStandbyMode()*
- *HAL_DBGMCU_DisableDBGStandbyMode()*

5.1.5 HAL SYSCFG configuration functions

This section provides functions allowing to:

- Start a hardware SRAM2 erase operation
- Enable/Disable the Internal FLASH Bank Swapping
- Configure the Voltage reference buffer
- Enable/Disable the Voltage reference buffer
- Enable/Disable the I/O analog switch voltage booster

This section contains the following APIs:

- *HAL_SYSCFG_SRAM2Erase()*
- *HAL_SYSCFG_EnableMemorySwappingBank()*
- *HAL_SYSCFG_DisableMemorySwappingBank()*
- *HAL_SYSCFG_VREFBUF_VoltageScalingConfig()*
- *HAL_SYSCFG_VREFBUF_HighImpedanceConfig()*
- *HAL_SYSCFG_VREFBUF_TrimmingConfig()*
- *HAL_SYSCFG_EnableVREFBUF()*
- *HAL_SYSCFG_DisableVREFBUF()*
- *HAL_SYSCFG_EnableIOAnalogSwitchBooster()*
- *HAL_SYSCFG_DisableIOAnalogSwitchBooster()*

5.1.6 Detailed description of functions

HAL_Init

Function name	**HAL_StatusTypeDef HAL_Init (void)**
Function description	Configure the Flash prefetch, the Instruction and Data caches, the time base source, NVIC and any required global low level hardware by calling the HAL_MspInit() callback function to be optionally defined in user file stm32l4xx_hal_msp.c.
Return values	• **HAL:** status
Notes	• HAL_Init() function is called at the beginning of program after reset and before the clock configuration. • In the default implementation the System Timer (Systick) is used as source of time base. The Systick configuration is based on MSI clock, as MSI is the clock used after a system Reset and the NVIC configuration is set to Priority group 4. Once done, time base tick starts incrementing: the tick variable counter is incremented each 1ms in the SysTick_Handler() interrupt handler.

HAL_DeInit

Function name	**HAL_StatusTypeDef HAL_DeInit (void)**
Function description	De-initialize common part of the HAL and stop the source of time base.
Return values	• **HAL:** status
Notes	• This function is optional.

HAL_MspInit

Function name	**void HAL_MspInit (void)**
Function description	Initialize the MSP.
Return values	• **None:**

HAL_MspDeInit

Function name	**void HAL_MspDeInit (void)**
Function description	DeInitialize the MSP.
Return values	• **None:**

HAL_InitTick

Function name	**HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)**
Function description	This function configures the source of the time base: The time source is configured to have 1ms time base with a dedicated Tick interrupt priority.
Parameters	• **TickPriority:** Tick interrupt priority.

Return values	• **HAL:** status
Notes	• This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig(). • In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, The SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked. The function is declared as __weak to be overwritten in case of other implementation in user file.

HAL_IncTick

Function name	**void HAL_IncTick (void)**
Function description	This function is called to increment a global variable "uwTick" used as application time base.
Return values	• **None:**
Notes	• In the default implementation, this variable is incremented each 1ms in SysTick ISR. • This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_Delay

Function name	**void HAL_Delay (uint32_t Delay)**
Function description	This function provides minimum delay (in milliseconds) based on variable incremented.
Parameters	• **Delay:** specifies the delay time length, in milliseconds.
Return values	• **None:**
Notes	• In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented. • This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTick

Function name	**uint32_t HAL_GetTick (void)**
Function description	Provide a tick value in millisecond.
Return values	• **tick:** value
Notes	• This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_SuspendTick

Function name	**void HAL_SuspendTick (void)**

Function description	Suspend Tick increment.
Return values	• **None:**
Notes	• In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the SysTick interrupt will be disabled and so Tick increment is suspended. • This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_ResumeTick

Function name	**void HAL_ResumeTick (void)**
Function description	Resume Tick increment.
Return values	• **None:**
Notes	• In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the SysTick interrupt will be enabled and so Tick increment is resumed. • This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetHalVersion

Function name	**uint32_t HAL_GetHalVersion (void)**
Function description	Return the HAL revision.
Return values	• **version:** 0xXYZR (8bits for each decimal, R for RC)

HAL_GetREVID

Function name	**uint32_t HAL_GetREVID (void)**
Function description	Return the device revision identifier.
Return values	• **Device:** revision identifier

HAL_GetDEVID

Function name	**uint32_t HAL_GetDEVID (void)**
Function description	Return the device identifier.
Return values	• **Device:** identifier

HAL_GetUIDw0

Function name	**uint32_t HAL_GetUIDw0 (void)**
Function description	Return the first word of the unique device identifier (UID based on 96 bits)
Return values	• **Device:** identifier

HAL_GetUIDw1

Function name	**uint32_t HAL_GetUIDw1 (void)**
Function description	Return the second word of the unique device identifier (UID based on 96 bits)
Return values	• **Device:** identifier

HAL_GetUIDw2

Function name	**uint32_t HAL_GetUIDw2 (void)**
Function description	Return the third word of the unique device identifier (UID based on 96 bits)
Return values	• **Device:** identifier

HAL_DBGMCU_EnableDBGSleepMode

Function name	**void HAL_DBGMCU_EnableDBGSleepMode (void)**
Function description	Enable the Debug Module during SLEEP mode.
Return values	• **None:**

HAL_DBGMCU_DisableDBGSleepMode

Function name	**void HAL_DBGMCU_DisableDBGSleepMode (void)**
Function description	Disable the Debug Module during SLEEP mode.
Return values	• **None:**

HAL_DBGMCU_EnableDBGStopMode

Function name	**void HAL_DBGMCU_EnableDBGStopMode (void)**
Function description	Enable the Debug Module during STOP0/STOP1/STOP2 modes.
Return values	• **None:**

HAL_DBGMCU_DisableDBGStopMode

Function name	**void HAL_DBGMCU_DisableDBGStopMode (void)**
Function description	Disable the Debug Module during STOP0/STOP1/STOP2 modes.
Return values	• **None:**

HAL_DBGMCU_EnableDBGStandbyMode

Function name	**void HAL_DBGMCU_EnableDBGStandbyMode (void)**
Function description	Enable the Debug Module during STANDBY mode.
Return values	• **None:**

HAL_DBGMCU_DisableDBGStandbyMode

Function name	**void HAL_DBGMCU_DisableDBGStandbyMode (void)**
Function description	Disable the Debug Module during STANDBY mode.
Return values	• **None:**

HAL_SYSCFG_SRAM2Erase

Function name	**void HAL_SYSCFG_SRAM2Erase (void)**
Function description	Start a hardware SRAM2 erase operation.
Return values	• **None:**
Notes	• As long as SRAM2 is not erased the SRAM2ER bit will be set. This bit is automatically reset at the end of the SRAM2 erase operation.

HAL_SYSCFG_EnableMemorySwappingBank

Function name	**void HAL_SYSCFG_EnableMemorySwappingBank (void)**
Function description	Enable the Internal FLASH Bank Swapping.
Return values	• **None:**
Notes	• This function can be used only for STM32L4xx devices. • Flash Bank2 mapped at 0x08000000 (and aliased @0x00000000) and Flash Bank1 mapped at 0x08100000 (and aliased at 0x00100000)

HAL_SYSCFG_DisableMemorySwappingBank

Function name	**void HAL_SYSCFG_DisableMemorySwappingBank (void)**
Function description	Disable the Internal FLASH Bank Swapping.
Return values	• **None:**
Notes	• This function can be used only for STM32L4xx devices. • The default state: Flash Bank1 mapped at 0x08000000 (and aliased @0x0000 0000) and Flash Bank2 mapped at 0x08100000 (and aliased at 0x00100000)

HAL_SYSCFG_VREFBUF_VoltageScalingConfig

Function name	**void HAL_SYSCFG_VREFBUF_VoltageScalingConfig (uint32_t VoltageScaling)**
Function description	Configure the internal voltage reference buffer voltage scale.
Parameters	• **VoltageScaling:** specifies the output voltage to achieve This parameter can be one of the following values: – SYSCFG_VREFBUF_VOLTAGE_SCALE0: VREF_OUT1 around 2.048 V. This requires VDDA equal to or higher than 2.4 V. – SYSCFG_VREFBUF_VOLTAGE_SCALE1: VREF_OUT2 around 2.5 V. This requires VDDA equal to

or higher than 2.8 V.

Return values	• **None:**

HAL_SYSCFG_VREFBUF_HighImpedanceConfig

Function name	**void HAL_SYSCFG_VREFBUF_HighImpedanceConfig (uint32_t Mode)**
Function description	Configure the internal voltage reference buffer high impedance mode.
Parameters	• **Mode:** specifies the high impedance mode This parameter can be one of the following values: – SYSCFG_VREFBUF_HIGH_IMPEDANCE_DISABLE: VREF+ pin is internally connect to VREFINT output. – SYSCFG_VREFBUF_HIGH_IMPEDANCE_ENABLE: VREF+ pin is high impedance.
Return values	• **None:**

HAL_SYSCFG_VREFBUF_TrimmingConfig

Function name	**void HAL_SYSCFG_VREFBUF_TrimmingConfig (uint32_t TrimmingValue)**
Function description	Tune the Internal Voltage Reference buffer (VREFBUF).
Return values	• **None:**

HAL_SYSCFG_EnableVREFBUF

Function name	**HAL_StatusTypeDef HAL_SYSCFG_EnableVREFBUF (void)**
Function description	Enable the Internal Voltage Reference buffer (VREFBUF).
Return values	• **HAL_OK/HAL_TIMEOUT:**

HAL_SYSCFG_DisableVREFBUF

Function name	**void HAL_SYSCFG_DisableVREFBUF (void)**
Function description	Disable the Internal Voltage Reference buffer (VREFBUF).
Return values	• **None:**

HAL_SYSCFG_EnableIOAnalogSwitchBooster

Function name	**void HAL_SYSCFG_EnableIOAnalogSwitchBooster (void)**
Function description	Enable the I/O analog switch voltage booster.
Return values	• **None:**

HAL_SYSCFG_DisableIOAnalogSwitchBooster

Function name	**void HAL_SYSCFG_DisableIOAnalogSwitchBooster (void)**
Function description	Disable the I/O analog switch voltage booster.

Return values • **None:**

5.2 HAL Firmware driver defines

5.2.1 HAL

DBGMCU Exported Macros

__HAL_DBGMCU_FREEZE_TIM2

__HAL_DBGMCU_UNFREEZE_TIM2

__HAL_DBGMCU_FREEZE_TIM3

__HAL_DBGMCU_UNFREEZE_TIM3

__HAL_DBGMCU_FREEZE_TIM4

__HAL_DBGMCU_UNFREEZE_TIM4

__HAL_DBGMCU_FREEZE_TIM5

__HAL_DBGMCU_UNFREEZE_TIM5

__HAL_DBGMCU_FREEZE_TIM6

__HAL_DBGMCU_UNFREEZE_TIM6

__HAL_DBGMCU_FREEZE_TIM7

__HAL_DBGMCU_UNFREEZE_TIM7

__HAL_DBGMCU_FREEZE_RTC

__HAL_DBGMCU_UNFREEZE_RTC

__HAL_DBGMCU_FREEZE_WWDG

__HAL_DBGMCU_UNFREEZE_WWDG

__HAL_DBGMCU_FREEZE_IWDG

__HAL_DBGMCU_UNFREEZE_IWDG

__HAL_DBGMCU_FREEZE_I2C1_TIMEOUT

__HAL_DBGMCU_UNFREEZE_I2C1_TIMEOUT

__HAL_DBGMCU_FREEZE_I2C2_TIMEOUT

__HAL_DBGMCU_UNFREEZE_I2C2_TIMEOUT

__HAL_DBGMCU_FREEZE_I2C3_TIMEOUT

__HAL_DBGMCU_UNFREEZE_I2C3_TIMEOUT

__HAL_DBGMCU_FREEZE_I2C4_TIMEOUT

__HAL_DBGMCU_UNFREEZE_I2C4_TIMEOUT

__HAL_DBGMCU_FREEZE_CAN1

__HAL_DBGMCU_UNFREEZE_CAN1

__HAL_DBGMCU_FREEZE_LPTIM1

__HAL_DBGMCU_UNFREEZE_LPTIM1

__HAL_DBGMCU_FREEZE_LPTIM2

__HAL_DBGMCU_UNFREEZE_LPTIM2

__HAL_DBGMCU_FREEZE_TIM1

__HAL_DBGMCU_UNFREEZE_TIM1

__HAL_DBGMCU_FREEZE_TIM8

__HAL_DBGMCU_UNFREEZE_TIM8

__HAL_DBGMCU_FREEZE_TIM15

__HAL_DBGMCU_UNFREEZE_TIM15

__HAL_DBGMCU_FREEZE_TIM16

__HAL_DBGMCU_UNFREEZE_TIM16

__HAL_DBGMCU_FREEZE_TIM17

__HAL_DBGMCU_UNFREEZE_TIM17

HAL state definition

HAL_SMBUS_STATE_RESET	SMBUS not yet initialized or disabled
HAL_SMBUS_STATE_READY	SMBUS initialized and ready for use
HAL_SMBUS_STATE_BUSY	SMBUS internal process is ongoing
HAL_SMBUS_STATE_MASTER_BUSY_TX	Master Data Transmission process is ongoing
HAL_SMBUS_STATE_MASTER_BUSY_RX	Master Data Reception process is ongoing
HAL_SMBUS_STATE_SLAVE_BUSY_TX	Slave Data Transmission process is ongoing
HAL_SMBUS_STATE_SLAVE_BUSY_RX	Slave Data Reception process is ongoing
HAL_SMBUS_STATE_TIMEOUT	Timeout state
HAL_SMBUS_STATE_ERROR	Reception process is ongoing
HAL_SMBUS_STATE_LISTEN	Address Listen Mode is ongoing

Boot Mode

SYSCFG_BOOT_MAINFLASH

SYSCFG_BOOT_SYSTEMFLASH

SYSCFG_BOOT_FMC

SYSCFG_BOOT_SRAM

SYSCFG_BOOT_OCTOPSPI1

SYSCFG_BOOT_OCTOPSPI2

SYSCFG Exported Macros

__HAL_SYSCFG_REMAPMEMORY_FLASH

__HAL_SYSCFG_REMAPMEMORY_SYSTEMFLASH

__HAL_SYSCFG_REMAPMEMORY_SRAM

__HAL_SYSCFG_REMAPMEMORY_FMC

__HAL_SYSCFG_REMAPMEMORY_OCTOSPI1

__HAL_SYSCFG_REMAPMEMORY_OCTOSPI2

__HAL_SYSCFG_GET_BOOT_MODE

Description:

- Return the boot mode as configured by user.

Return value:

- The: boot mode as configured by user. The returned value can be one of the following values:
  - SYSCFG_BOOT_MAINFLASH
  - SYSCFG_BOOT_SYSTEMFLASH
  - SYSCFG_BOOT_SRAM
  - SYSCFG_BOOT_QUADSPI

__HAL_SYSCFG_SRAM2_WRP_1_31_ENABLE

Description:

- SRAM2 page 0 to 31 write protection enable macro.

Parameters:

- __SRAM2WRP__: This parameter can be a combination of values of

Notes:

- Write protection can only be disabled by a system reset

__HAL_SYSCFG_SRAM2_WRP_32_63_ENABLE

Description:

- SRAM2 page 32 to 63 write protection enable macro.

Parameters:

- __SRAM2WRP__: This parameter can be a combination of values of

Notes:

- Write protection can only be disabled by a system reset

__HAL_SYSCFG_SRAM2_WRP_UNLOCK

Notes:

- Writing a wrong key reactivates the write protection

__HAL_SYSCFG_SRAM2_ERASE

Notes:

- __SYSCFG_GET_FLAG(SYSCFG_FLAG_SRAM2_BUSY) may be used to check end of erase

__HAL_SYSCFG_FPU_INTERRUPT_ENABLE	**Description:** • Floating Point Unit interrupt enable/disable macros. **Parameters:** • __INTERRUPT__: This parameter can be a value of
__HAL_SYSCFG_FPU_INTERRUPT_DISABLE	
__HAL_SYSCFG_BREAK_ECC_LOCK	**Notes:** • The selected configuration is locked and can be unlocked only by system reset. Enable and lock the connection of Flash ECC error connection to TIM1/8/15/16/17 Break input.
__HAL_SYSCFG_BREAK_LOCKUP_LOCK	**Notes:** • The selected configuration is locked and can be unlocked only by system reset. Enable and lock the connection of Cortex-M4 LOCKUP (Hardfault) output to TIM1/8/15/16/17 Break input.
__HAL_SYSCFG_BREAK_PVD_LOCK	**Notes:** • The selected configuration is locked and can be unlocked only by system reset. Enable and lock the PVD connection to Timer1/8/15/16/17 Break input, as well as the PVDE and PLS[2:0] in the PWR_CR2 register.
__HAL_SYSCFG_BREAK_SRAM2PARITY_LOCK	**Notes:** • The selected configuration is locked and can be unlocked by system reset. Enable and lock the SRAM2 parity error signal connection to TIM1/8/15/16/17 Break input.
__HAL_SYSCFG_GET_FLAG	**Description:** • Check SYSCFG flag is set or not. **Parameters:** • __FLAG__: specifies the flag to check. This parameter can be one of the following values: – SYSCFG_FLAG_SRAM2_PE SRAM2 Parity Error Flag – SYSCFG_FLAG_SRAM2_BUSY SRAM2 Erase Ongoing **Return value:** • The: new state of __FLAG__ (TRUE or

FALSE).

__HAL_SYSCFG_CLEAR_FLAG

__HAL_SYSCFG_FASTMODEPLUS_ENABLE

Description:

- Fast-mode Plus driving capability enable/disable macros.

Parameters:

- __FASTMODEPLUS__: This parameter can be a value of:
  - SYSCFG_FASTMODEPLUS_PB6 Fast-mode Plus driving capability activation on PB6
  - SYSCFG_FASTMODEPLUS_PB7 Fast-mode Plus driving capability activation on PB7
  - SYSCFG_FASTMODEPLUS_PB8 Fast-mode Plus driving capability activation on PB8
  - SYSCFG_FASTMODEPLUS_PB9 Fast-mode Plus driving capability activation on PB9

__HAL_SYSCFG_FASTMODEPLUS_DISABLE

Fast-mode Plus on GPIO

SYSCFG_FASTMODEPLUS_PB6	Enable Fast-mode Plus on PB6
SYSCFG_FASTMODEPLUS_PB7	Enable Fast-mode Plus on PB7
SYSCFG_FASTMODEPLUS_PB8	Enable Fast-mode Plus on PB8
SYSCFG_FASTMODEPLUS_PB9	Enable Fast-mode Plus on PB9

Flags

SYSCFG_FLAG_SRAM2_PE	SRAM2 parity error
SYSCFG_FLAG_SRAM2_BUSY	SRAM2 busy by erase operation

FPU Interrupts

SYSCFG_IT_FPU_IOC	Floating Point Unit Invalid operation Interrupt
SYSCFG_IT_FPU_DZC	Floating Point Unit Divide-by-zero Interrupt
SYSCFG_IT_FPU_UFC	Floating Point Unit Underflow Interrupt
SYSCFG_IT_FPU_OFC	Floating Point Unit Overflow Interrupt
SYSCFG_IT_FPU_IDC	Floating Point Unit Input denormal Interrupt
SYSCFG_IT_FPU_IXC	Floating Point Unit Inexact Interrupt

SRAM2 Page Write protection (0 to 31)

SYSCFG_SRAM2WRP_PAGE0	SRAM2 Write protection page 0
SYSCFG_SRAM2WRP_PAGE1	SRAM2 Write protection page 1
SYSCFG_SRAM2WRP_PAGE2	SRAM2 Write protection page 2

SYSCFG_SRAM2WRP_PAGE3	SRAM2 Write protection page 3
SYSCFG_SRAM2WRP_PAGE4	SRAM2 Write protection page 4
SYSCFG_SRAM2WRP_PAGE5	SRAM2 Write protection page 5
SYSCFG_SRAM2WRP_PAGE6	SRAM2 Write protection page 6
SYSCFG_SRAM2WRP_PAGE7	SRAM2 Write protection page 7
SYSCFG_SRAM2WRP_PAGE8	SRAM2 Write protection page 8
SYSCFG_SRAM2WRP_PAGE9	SRAM2 Write protection page 9
SYSCFG_SRAM2WRP_PAGE10	SRAM2 Write protection page 10
SYSCFG_SRAM2WRP_PAGE11	SRAM2 Write protection page 11
SYSCFG_SRAM2WRP_PAGE12	SRAM2 Write protection page 12
SYSCFG_SRAM2WRP_PAGE13	SRAM2 Write protection page 13
SYSCFG_SRAM2WRP_PAGE14	SRAM2 Write protection page 14
SYSCFG_SRAM2WRP_PAGE15	SRAM2 Write protection page 15
SYSCFG_SRAM2WRP_PAGE16	SRAM2 Write protection page 16
SYSCFG_SRAM2WRP_PAGE17	SRAM2 Write protection page 17
SYSCFG_SRAM2WRP_PAGE18	SRAM2 Write protection page 18
SYSCFG_SRAM2WRP_PAGE19	SRAM2 Write protection page 19
SYSCFG_SRAM2WRP_PAGE20	SRAM2 Write protection page 20
SYSCFG_SRAM2WRP_PAGE21	SRAM2 Write protection page 21
SYSCFG_SRAM2WRP_PAGE22	SRAM2 Write protection page 22
SYSCFG_SRAM2WRP_PAGE23	SRAM2 Write protection page 23
SYSCFG_SRAM2WRP_PAGE24	SRAM2 Write protection page 24
SYSCFG_SRAM2WRP_PAGE25	SRAM2 Write protection page 25
SYSCFG_SRAM2WRP_PAGE26	SRAM2 Write protection page 26
SYSCFG_SRAM2WRP_PAGE27	SRAM2 Write protection page 27
SYSCFG_SRAM2WRP_PAGE28	SRAM2 Write protection page 28
SYSCFG_SRAM2WRP_PAGE29	SRAM2 Write protection page 29
SYSCFG_SRAM2WRP_PAGE30	SRAM2 Write protection page 30
SYSCFG_SRAM2WRP_PAGE31	SRAM2 Write protection page 31

SRAM2 Page Write protection (32 to 63)

SYSCFG_SRAM2WRP_PAGE32	SRAM2 Write protection page 32
SYSCFG_SRAM2WRP_PAGE33	SRAM2 Write protection page 33
SYSCFG_SRAM2WRP_PAGE34	SRAM2 Write protection page 34
SYSCFG_SRAM2WRP_PAGE35	SRAM2 Write protection page 35
SYSCFG_SRAM2WRP_PAGE36	SRAM2 Write protection page 36
SYSCFG_SRAM2WRP_PAGE37	SRAM2 Write protection page 37

SYSCFG_SRAM2WRP_PAGE38	SRAM2 Write protection page 38
SYSCFG_SRAM2WRP_PAGE39	SRAM2 Write protection page 39
SYSCFG_SRAM2WRP_PAGE40	SRAM2 Write protection page 40
SYSCFG_SRAM2WRP_PAGE41	SRAM2 Write protection page 41
SYSCFG_SRAM2WRP_PAGE42	SRAM2 Write protection page 42
SYSCFG_SRAM2WRP_PAGE43	SRAM2 Write protection page 43
SYSCFG_SRAM2WRP_PAGE44	SRAM2 Write protection page 44
SYSCFG_SRAM2WRP_PAGE45	SRAM2 Write protection page 45
SYSCFG_SRAM2WRP_PAGE46	SRAM2 Write protection page 46
SYSCFG_SRAM2WRP_PAGE47	SRAM2 Write protection page 47
SYSCFG_SRAM2WRP_PAGE48	SRAM2 Write protection page 48
SYSCFG_SRAM2WRP_PAGE49	SRAM2 Write protection page 49
SYSCFG_SRAM2WRP_PAGE50	SRAM2 Write protection page 50
SYSCFG_SRAM2WRP_PAGE51	SRAM2 Write protection page 51
SYSCFG_SRAM2WRP_PAGE52	SRAM2 Write protection page 52
SYSCFG_SRAM2WRP_PAGE53	SRAM2 Write protection page 53
SYSCFG_SRAM2WRP_PAGE54	SRAM2 Write protection page 54
SYSCFG_SRAM2WRP_PAGE55	SRAM2 Write protection page 55
SYSCFG_SRAM2WRP_PAGE56	SRAM2 Write protection page 56
SYSCFG_SRAM2WRP_PAGE57	SRAM2 Write protection page 57
SYSCFG_SRAM2WRP_PAGE58	SRAM2 Write protection page 58
SYSCFG_SRAM2WRP_PAGE59	SRAM2 Write protection page 59
SYSCFG_SRAM2WRP_PAGE60	SRAM2 Write protection page 60
SYSCFG_SRAM2WRP_PAGE61	SRAM2 Write protection page 61
SYSCFG_SRAM2WRP_PAGE62	SRAM2 Write protection page 62
SYSCFG_SRAM2WRP_PAGE63	SRAM2 Write protection page 63

VREFBUF High Impedance

SYSCFG_VREFBUF_HIGH_IMPEDANCE_DISABLE	VREF_plus pin is internally connected to Voltage reference buffer output
SYSCFG_VREFBUF_HIGH_IMPEDANCE_ENABLE	VREF_plus pin is high impedance

VREFBUF Voltage Scale

SYSCFG_VREFBUF_VOLTAGE_SCALE0	Voltage reference scale 0 (VREF_OUT1)
SYSCFG_VREFBUF_VOLTAGE_SCALE1	Voltage reference scale 1 (VREF_OUT2)

6 HAL ADC Generic Driver

6.1 ADC Firmware driver registers structures

6.1.1 ADC_OversamplingTypeDef

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***
- ***uint32_t TriggeredMode***
- ***uint32_t OversamplingStopReset***

Field Documentation

- ***uint32_t ADC_OversamplingTypeDef::Ratio***
  Configures the oversampling ratio. This parameter can be a value of ***ADC_Oversampling_Ratio***
- ***uint32_t ADC_OversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_Right_Bit_Shift***
- ***uint32_t ADC_OversamplingTypeDef::TriggeredMode***
  Selects the regular triggered oversampling mode. This parameter can be a value of ***ADC_Triggered_Oversampling_Mode***
- ***uint32_t ADC_OversamplingTypeDef::OversamplingStopReset***
  Selects the regular oversampling mode. The oversampling is either temporary stopped or reset upon an injected sequence interruption. If oversampling is enabled on both regular and injected groups, this parameter is discarded and forced to setting "ADC_REGOVERSAMPLING_RESUMED_MODE" (the oversampling buffer is zeroed during injection sequence). This parameter can be a value of ***ADC_Regular_Oversampling_Mode***

6.1.2 ADC_InitTypeDef

Data Fields

- ***uint32_t ClockPrescaler***
- ***uint32_t Resolution***
- ***uint32_t DataAlign***
- ***uint32_t ScanConvMode***
- ***uint32_t EOCSelection***
- ***uint32_t LowPowerAutoWait***
- ***uint32_t ContinuousConvMode***
- ***uint32_t NbrOfConversion***
- ***uint32_t DiscontinuousConvMode***
- ***uint32_t NbrOfDiscConversion***
- ***uint32_t ExternalTrigConv***
- ***uint32_t ExternalTrigConvEdge***
- ***uint32_t DMAContinuousRequests***
- ***uint32_t Overrun***
- ***uint32_t OversamplingMode***
- ***ADC_OversamplingTypeDef Oversampling***
- ***uint32_t DFSDMConfig***

Field Documentation

- ***uint32_t ADC_InitTypeDef::ClockPrescaler***
  Select ADC clock source (synchronous clock derived from APB clock or asynchronous clock derived from system clock or PLL (Refer to reference manual for list of clocks available)) and clock prescaler. This parameter can be a value of ***ADC_ClockPrescaler***. Note: The ADC clock configuration is common to all ADC instances. Note: In case of usage of channels on injected group, ADC frequency should be lower than AHB clock frequency /4 for resolution 12 or 10 bits, AHB clock frequency /3 for resolution 8 bits, AHB clock frequency /2 for resolution 6 bits. Note: In case of synchronous clock mode based on HCLK/1, the configuration must be enabled only if the system clock has a 50% duty clock cycle (APB prescaler configured inside RCC must be bypassed and PCLK clock must have 50% duty cycle). Refer to reference manual for details. Note: In case of usage of asynchronous clock, the selected clock must be preliminarily enabled at RCC top level. Note: This parameter can be modified only if all ADC instances are disabled.
- ***uint32_t ADC_InitTypeDef::Resolution***
  Configure the ADC resolution. This parameter can be a value of ***ADC_Resolution***
- ***uint32_t ADC_InitTypeDef::DataAlign***
  Specify ADC data alignment in conversion data register (right or left). Refer to reference manual for alignments formats versus resolutions. This parameter can be a value of ***ADC_Data_align***
- ***uint32_t ADC_InitTypeDef::ScanConvMode***
  Configure the sequencer of ADC groups regular and injected. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by 'NbrOfConversion' or 'InjectedNbrOfConversion' and rank of each channel in sequencer). Scan direction is upward: from rank 1 to rank 'n'. This parameter can be a value of ***ADC_Scan_mode***
- ***uint32_t ADC_InitTypeDef::EOCSelection***
  Specify which EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of unitary conversion or end of sequence conversions. This parameter can be a value of ***ADC_EOCSelection***.
- ***uint32_t ADC_InitTypeDef::LowPowerAutoWait***
  Select the dynamic low power Auto Delay: new conversion start only when the previous conversion (for ADC group regular) or previous sequence (for ADC group injected) has been retrieved by user software, using function **HAL_ADC_GetValue()** or **HAL_ADCEx_InjectedGetValue()**. This feature automatically adapts the frequency of ADC conversions triggers to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: Do not use with interruption or DMA (**HAL_ADC_Start_IT()**, **HAL_ADC_Start_DMA()**) since they clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion with **HAL_ADC_Start()**, 2. Later on, when ADC conversion data is needed: use **HAL_ADC_PollForConversion()** to ensure that conversion is completed and **HAL_ADC_GetValue()** to retrieve conversion result and trig another conversion start. (in case of usage of ADC group injected, use the equivalent functions HAL_ADCExInjected_Start(), **HAL_ADCEx_InjectedGetValue()**, ...).
- ***uint32_t ADC_InitTypeDef::ContinuousConvMode***
  Specify whether the conversion is performed in single mode (one conversion) or continuous mode for ADC group regular, after the first ADC conversion start trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.

- ***uint32_t ADC_InitTypeDef::NbrOfConversion***
  Specify the number of ranks that will be converted within the regular group sequencer. To use the regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 16. Note: This parameter must be modified when no conversion is on going on regular group (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_InitTypeDef::DiscontinuousConvMode***
  Specify whether the conversions sequence of ADC group regular is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfDiscConversion***
  Specifies the number of discontinuous conversions in which the main sequence of ADC group regular (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConv***
  Select the external event source used to trigger ADC group regular conversion start. If set to ADC_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_regular_external_trigger_source***. Caution: external trigger source is common to all ADC instances.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConvEdge***
  Select the external event edge used to trigger ADC group regular conversion start. If trigger source is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of ***ADC_regular_external_trigger_edge***
- ***uint32_t ADC_InitTypeDef::DMAContinuousRequests***
  Specify whether the DMA requests are performed in one shot mode (DMA transfer stops when number of conversions is reached) or in continuous mode (DMA transfer unlimited, whatever number of conversions). This parameter can be set to ENABLE or DISABLE. Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached.
- ***uint32_t ADC_InitTypeDef::Overrun***
  Select the behavior in case of overrun: data overwritten or preserved (default). This parameter applies to ADC group regular only. This parameter can be a value of ***ADC_Overrun***. Note: In case of overrun set to data preserved and usage with programming model with interruption (HAL_Start_IT()): ADC IRQ handler has to clear end of conversion flags, this induces the release of the preserved data. If needed, this data can be saved in function **HAL_ADC_ConvCpltCallback()**, placed in user program code (called before end of conversion flags clear). Note: Error reporting with respect to the conversion mode:Usage with ADC conversion by polling for event or interruption: Error is reported only if overrun is set to data preserved. If overrun is set to data overwritten, user can willingly not read all the converted data, this is not considered as an erroneous case.Usage with ADC conversion by DMA: Error is reported whatever overrun setting (DMA is expected to process all data from data register).
- ***uint32_t ADC_InitTypeDef::OversamplingMode***
  Specify whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing on ADC groups regular and injected

- ***ADC_OversamplingTypeDef ADC_InitTypeDef::Oversampling***
  Specify the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling is already enabled.
- ***uint32_t ADC_InitTypeDef::DFSDMConfig***
  Specify whether ADC conversion data is sent directly to DFSDM. This parameter can be a value of ***ADCEx_DFSDM_Mode_Configuration***. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).

6.1.3 ADC_ChannelConfTypeDef

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t SamplingTime***
- ***uint32_t SingleDiff***
- ***uint32_t OffsetNumber***
- ***uint32_t Offset***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***
  Specify the channel to configure into ADC regular group. This parameter can be a value of ***ADC_channels*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_ChannelConfTypeDef::Rank***
  Specify the rank in the regular group sequencer. This parameter can be a value of ***ADC_regular_rank*** Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)
- ***uint32_t ADC_ChannelConfTypeDef::SamplingTime***
  Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME*** Caution: This parameter applies to a channel that can be used into regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values.
- ***uint32_t ADC_ChannelConfTypeDef::SingleDiff***
  Select single-ended or differential input. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of ***ADCEx_SingleDifferential*** Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)

- ***uint32_t ADC_ChannelConfTypeDef::OffsetNumber***
  Select the offset number This parameter can be a value of ***ADCEx_OffsetNumber*** Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.
- ***uint32_t ADC_ChannelConfTypeDef::Offset***
  Define the offset to be subtracted from the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

6.1.4 ADC_AnalogWDGConfTypeDef

Data Fields

- ***uint32_t WatchdogNumber***
- ***uint32_t WatchdogMode***
- ***uint32_t Channel***
- ***uint32_t ITMode***
- ***uint32_t HighThreshold***
- ***uint32_t LowThreshold***

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber***
  Select which ADC analog watchdog is monitoring the selected channel. For Analog Watchdog 1: Only 1 channel can be monitored (or overall group of channels by setting parameter 'WatchdogMode') For Analog Watchdog 2 and 3: Several channels can be monitored (by successive calls of '**HAL_ADC_AnalogWDGConfig()**' for each channel) This parameter can be a value of ***ADC_analog_watchdog_number***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode***
  Configure the ADC analog watchdog mode: single/all/none channels. For Analog Watchdog 1: Configure the ADC analog watchdog mode: single channel or all channels, ADC groups regular and-or injected.For Analog Watchdog 2 and 3: There is no configuration for all channels as AWD1. Set value 'ADC_ANALOGWATCHDOG_NONE' to reset channels group programmed with parameter 'Channel', set any other value to program the channel(s) to be monitored. This parameter can be a value of ***ADC_analog_watchdog_mode***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel***
  Select which ADC channel to monitor by analog watchdog. For Analog Watchdog 1: this parameter has an effect only if parameter 'WatchdogMode' is configured on single channel (only 1 channel can be monitored). For Analog Watchdog 2 and 3: Several channels can be monitored. To use this feature, call successively the function **HAL_ADC_AnalogWDGConfig()** for each channel to be added (or removed with value 'ADC_ANALOGWATCHDOG_NONE'). This parameter can be a value of ***ADC_channels***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::ITMode***
  Specify whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold***
  Configure the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.

- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***
  Configures the ADC analog watchdog Low threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.

6.1.5 ADC_InjectionConfigTypeDef

Data Fields

- ***uint32_t ContextQueue***
- ***uint32_t ChannelCount***

Field Documentation

- ***uint32_t ADC_InjectionConfigTypeDef::ContextQueue***
  Injected channel configuration context: build-up over each **HAL_ADCEx_InjectedConfigChannel()** call to finally initialize JSQR register at **HAL_ADCEx_InjectedConfigChannel()** last call
- ***uint32_t ADC_InjectionConfigTypeDef::ChannelCount***
  Number of channels in the injected sequence

6.1.6 ADC_HandleTypeDef

Data Fields

- ***ADC_TypeDef * Instance***
- ***ADC_InitTypeDef Init***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***ADC_InjectionConfigTypeDef InjectionConfig***

Field Documentation

- ***ADC_TypeDef* ADC_HandleTypeDef::Instance***
  Register base address
- ***ADC_InitTypeDef ADC_HandleTypeDef::Init***
  ADC initialization parameters and regular conversions setting
- ***DMA_HandleTypeDef* ADC_HandleTypeDef::DMA_Handle***
  Pointer DMA Handler
- ***HAL_LockTypeDef ADC_HandleTypeDef::Lock***
  ADC locking object
- ***__IO uint32_t ADC_HandleTypeDef::State***
  ADC communication state (bitmap of ADC states)
- ***__IO uint32_t ADC_HandleTypeDef::ErrorCode***
  ADC Error code
- ***ADC_InjectionConfigTypeDef ADC_HandleTypeDef::InjectionConfig***
  ADC injected channel configuration build-up structure

6.2 ADC Firmware driver API description

6.2.1 ADC peripheral features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution.

- Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.
- Programmable sampling time (channel wise)
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- Configurable delay between conversions in Dual interleaved mode.
- ADC channels selectable single/differential input.
- ADC offset shared on 4 offset instances.
- ADC calibration
- ADC conversion of regular group.
- ADC supply requirements: 1.62 V to 3.6 V.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

6.2.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
   - As prerequisite, ADC clock must be configured at RCC top level.
   - Two clock settings are mandatory:
     - ADC clock (core clock, also possibly conversion clock).
     - ADC clock (conversions clock). Two possible clock sources: synchronous clock derived from APB clock or asynchronous clock derived from system clock, PLLSAI1 or the PLLSAI2 running up to 80MHz.
     - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC_CLK_ENABLE(); (mandatory) RCC_ADCCLKSOURCE_PLL enable: (optional: if asynchronous clock selected)
     - RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
     - PeriphClkInit.AdcClockSelection = RCC_ADCCLKSOURCE_PLL;
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
   - ADC clock source and clock prescaler are configured at ADC level with parameter "ClockPrescaler" using function HAL_ADC_Init().
2. ADC pins configuration
   - Enable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_ENABLE()
   - Configure these ADC pins in analog mode using function HAL_GPIO_Init()
3. Optionally, in case of usage of ADC with interruptions:
   - Configure the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding ADC interruption vector ADCx_IRQHandler().
4. Optionally, in case of usage of DMA:
   - Configure the DMA (DMA channel, mode normal or circular, ...) using function HAL_DMA_Init().
   - Configure the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

  - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding DMA interruption vector DMAx_Channelx_IRQHandler().

Configuration of ADC, group regular, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ...) using function HAL_ADC_Init().
2. Configure the channels for regular group parameters (channel number, channel rank into sequencer, ..., into regular group) using function HAL_ADC_ConfigChannel().
3. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function HAL_ADC_AnalogWDGConfig().

Execution of ADC conversions

1. Optionally, perform an automatic ADC calibration to improve the conversion accuracy using function HAL_ADCEx_Calibration_Start().
2. ADC driver can be used among three modes: polling, interruption, transfer by DMA.
   - ADC conversion by polling:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start()
     - Wait for ADC conversion completion using function HAL_ADC_PollForConversion()
     - Retrieve conversion results using function HAL_ADC_GetValue()
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop()
   - ADC conversion by interruption:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_IT()
     - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() (this function must be implemented in user program)
     - Retrieve conversion results using function HAL_ADC_GetValue()
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_IT()
   - ADC conversion with transfer by DMA:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_DMA()
     - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() or HAL_ADC_ConvHalfCpltCallback() (these functions must be implemented in user program)
     - Conversion results are automatically transferred by DMA into destination variable address.
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_DMA()

Callback functions must be implemented in user program:

- HAL_ADC_ErrorCallback()
- HAL_ADC_LevelOutOfWindowCallback() (callback of analog watchdog)
- HAL_ADC_ConvCpltCallback()
- HAL_ADC_ConvHalfCpltCallback

Deinitialization of ADC

1. Disable the ADC interface
   - ADC clock can be hard reset and disabled at RCC top level.
   - Hard reset of ADC peripherals using macro __ADCx_FORCE_RESET(), __ADCx_RELEASE_RESET().
   - ADC clock disable using the equivalent macro/functions as configuration step.
     - Example: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
     - RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI14;
     - RCC_OscInitStructure.HSI14State = RCC_HSI14_OFF; (if not used for system clock)
     - HAL_RCC_OscConfig(&RCC_OscInitStructure);
2. ADC pins configuration
   - Disable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
   - Disable the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
4. Optionally, in case of usage of DMA:
   - Deinitialize the DMA using function HAL_DMA_Init().
   - Disable the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

6.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog

This section contains the following APIs:

- ***HAL_ADC_ConfigChannel()***
- ***HAL_ADC_AnalogWDGConfig()***

6.2.4 Peripheral state and errors functions

This subsection provides functions to get in run-time the status of the peripheral.

- Check the ADC state
- Check the ADC error code

This section contains the following APIs:

- ***HAL_ADC_GetState()***
- ***HAL_ADC_GetError()***

6.2.5 Detailed description of functions

HAL_ADC_Init

Function name	**HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)**
Function description	Initialize the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".
Parameters	• **hadc:** ADC handle

Return values	• **HAL:** status
Notes	• As prerequisite, ADC clock must be configured at RCC top level (refer to description of RCC configuration for ADC in header of this file). • Possibility to update parameters on the fly: This function initializes the ADC MSP (HAL_ADC_MspInit()) only when coming from ADC state reset. Following calls to this function can be used to reconfigure some parameters of ADC_InitTypeDef structure on the fly, without modifying MSP configuration. If ADC MSP has to be modified again, HAL_ADC_DeInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef". • This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef". • Parameters related to common ADC registers (ADC clock mode) are set only if all ADCs are disabled. If this is not the case, these common parameters setting are bypassed without error reporting: it can be the intended behaviour in case of update of a parameter of ADC_InitTypeDef on the fly, without disabling the other ADCs.

HAL_ADC_DeInit

Function name	**HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)**
Function description	Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status
Notes	• For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common group are disabled. (function "HAL_ADC_MspDeInit()" is also called under the same conditions: all ADC instances use the same core clock at RCC level, disabling the core clock reset all ADC instances). If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behavior in case of reset of a single ADC while the other ADCs sharing the same common group is still running. • By default, HAL_ADC_DeInit() set ADC in mode deep power-down: this saves more power by reducing leakage currents and is particularly interesting before entering MCU low-power modes.

HAL_ADC_MspInit

Function name	**void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)**
Function description	Initialize the ADC MSP.

Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADC_MspDeInit

Function name	**void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)**
Function description	DeInitialize the ADC MSP.
Parameters	• **hadc:** ADC handle
Return values	• **None:**
Notes	• All ADC instances use the same core clock at RCC level, disabling the core clock reset all ADC instances).

HAL_ADC_Start

Function name	**HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)**
Function description	Enable ADC, start conversion of regular group.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status
Notes	• Interruptions enabled in this function: None. • Case of multimode enabled (when multimode feature is available): if ADC is Slave, ADC is enabled but conversion is not started, if ADC is master, ADC is enabled and multimode conversion is started.

HAL_ADC_Stop

Function name	**HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)**
Function description	Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.
Notes	• : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.

HAL_ADC_PollForConversion

Function name	**HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)**
Function description	Wait for regular group conversion to be completed.
Parameters	• **hadc:** ADC handle • **Timeout:** Timeout value in millisecond.

Return values	• **HAL:** status
Notes	• ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function, with an exception: if low power feature "LowPowerAutoWait" is enabled, flags are not cleared to not interfere with this feature until data register is read using function HAL_ADC_GetValue(). • This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each conversion (ADC init parameter "EOCSelection" set to ADC_EOC_SINGLE_CONV). In this case, DMA resets the flag EOC and polling cannot be performed on each conversion. Nevertheless, polling can still be performed on the complete sequence (ADC init parameter "EOCSelection" set to ADC_EOC_SEQ_CONV).

HAL_ADC_PollForEvent

Function name	**HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)**
Function description	Poll for ADC event.
Parameters	• **hadc:** ADC handle • **EventType:** the ADC event type. This parameter can be one of the following values: – ADC_EOSMP_EVENT ADC End of Sampling event – ADC_AWD1_EVENT ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 devices) – ADC_AWD2_EVENT ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 families) – ADC_AWD3_EVENT ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 families) – ADC_OVR_EVENT ADC Overrun event – ADC_JQOVF_EVENT ADC Injected context queue overflow event • **Timeout:** Timeout value in millisecond.
Return values	• **HAL:** status
Notes	• The relevant flag is cleared if found to be set, except for ADC_FLAG_OVR. Indeed, the latter is reset only if hadc->Init.Overrun field is set to ADC_OVR_DATA_OVERWRITTEN. Otherwise, data register may be potentially overwritten by a new converted data as soon as OVR is cleared. To reset OVR flag once the preserved data is retrieved, the user can resort to macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_OVR);

HAL_ADC_Start_IT

Function name	**HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)**

Function description	Enable ADC, start conversion of regular group with interruption.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status
Notes	• Interruptions enabled in this function according to initialization setting: EOC (end of conversion), EOS (end of sequence), OVR overrun. Each of these interruptions has its dedicated callback function. • Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_IT() must be called for ADC Slave first, then for ADC Master. For ADC Slave, ADC is enabled only (conversion is not started). For ADC Master, ADC is enabled and multimode conversion is started. • To guarantee a proper reset of all interruptions once all the needed conversions are obtained, HAL_ADC_Stop_IT() must be called to ensure a correct stop of the IT-based conversions. • By default, HAL_ADC_Start_IT() does not enable the End Of Sampling interruption. If required (e.g. in case of oversampling with trigger mode), the user must: 1. first clear the EOSMP flag if set with macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_EOSMP) 2. then enable the EOSMP interrupt with macro __HAL_ADC_ENABLE_IT(hadc, ADC_IT_EOSMP) before calling HAL_ADC_Start_IT().

HAL_ADC_Stop_IT

Function name	**HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)**
Function description	Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interrution of end-of-conversion, disable ADC peripheral.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.

HAL_ADC_Start_DMA

Function name	**HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)**
Function description	Enable ADC, start conversion of regular group and transfer result through DMA.
Parameters	• **hadc:** ADC handle • **pData:** Destination Buffer address. • **Length:** Length of data to be transferred from ADC peripheral to memory (in bytes)
Return values	• **HAL:** status.
Notes	• Interruptions enabled in this function: overrun (if applicable), DMA half transfer, DMA transfer complete. Each of these

interruptions has its dedicated callback function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_DMA() is designed for single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStart_DMA() function must be used.

HAL_ADC_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)**
Function description	Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.
Notes	• : ADC peripheral disable is forcing stop of potential conversion on ADC group injected. If ADC group injected is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function. • Case of multimode enabled (when multimode feature is available): HAL_ADC_Stop_DMA() function is dedicated to single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStop_DMA() API must be used.

HAL_ADC_GetValue

Function name	**uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)**
Function description	Get ADC regular group conversion result.
Parameters	• **hadc:** ADC handle
Return values	• **ADC:** group regular conversion data
Notes	• Reading register DR automatically clears ADC flag EOC (ADC group regular end of unitary conversion). • This function does not clear ADC flag EOS (ADC group regular end of sequence conversion). Occurrence of flag EOS rising: If sequencer is composed of 1 rank, flag EOS is equivalent to flag EOC.If sequencer is composed of several ranks, during the scan sequence flag EOC only is raised, at the end of the scan sequence both flags EOC and EOS are raised. To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADC_PollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS).

HAL_ADC_IRQHandler

Function name	**void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)**
Function description	Handle ADC interrupt request.
Parameters	• **hadc:** ADC handle

Return values	• **None:**

HAL_ADC_ConvCpltCallback

Function name	**void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)**
Function description	Conversion complete callback in non-blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADC_ConvHalfCpltCallback

Function name	**void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)**
Function description	Conversion DMA half-transfer callback in non-blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADC_LevelOutOfWindowCallback

Function name	**void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)**
Function description	Analog watchdog 1 callback in non-blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADC_ErrorCallback

Function name	**void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)**
Function description	ADC error callback in non-blocking mode (ADC conversion with interruption or transfer by DMA).
Parameters	• **hadc:** ADC handle
Return values	• **None:**
Notes	• In case of error due to overrun when using ADC with DMA transfer (HAL ADC handle paramater "ErrorCode" to state "HAL_ADC_ERROR_OVR"): Reinitialize the DMA using function "HAL_ADC_Stop_DMA()".If needed, restart a new ADC conversion using function "HAL_ADC_Start_DMA()" (this function is also clearing overrun flag)

HAL_ADC_ConfigChannel

Function name	**HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)**
Function description	Configure a channel to be assigned to ADC group regular.

Parameters	• **hadc:** ADC handle • **sConfig:** Structure of ADC channel assigned to ADC group regular.
Return values	• **HAL:** status
Notes	• In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit(). • Possibility to update parameters on the fly: This function initializes channel into ADC group regular, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_ChannelConfTypeDef".

HAL_ADC_AnalogWDGConfig

Function name	**HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)**
Function description	Configure the analog watchdog.
Parameters	• **hadc:** ADC handle • **AnalogWDGConfig:** Structure of ADC analog watchdog configuration
Return values	• **HAL:** status
Notes	• Possibility to update parameters on the fly: This function initializes the selected analog watchdog, successive calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_AnalogWDGConfTypeDef". • On this STM32 serie, analog watchdog thresholds cannot be modified while ADC conversion is on going.

HAL_ADC_GetState

Function name	**uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)**
Function description	Return the ADC handle state.
Parameters	• **hadc:** ADC handle
Return values	• **ADC:** handle state (bitfield on 32 bits)
Notes	• ADC state machine is managed by bitfields, ADC status must be compared with states bits. For example: " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_REG_BUSY)) " " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_AWD1)) "

HAL_ADC_GetError

Function name	**uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)**
Function description	Return the ADC error code.
Parameters	• **hadc:** ADC handle
Return values	• **ADC:** error code (bitfield on 32 bits)

ADC_ConversionStop

Function name	**HAL_StatusTypeDef ADC_ConversionStop (ADC_HandleTypeDef * hadc, uint32_t ConversionGroup)**
Function description	Stop ADC conversion.
Parameters	• **hadc:** ADC handle • **ConversionGroup:** ADC group regular and/or injected. This parameter can be one of the following values: – ADC_REGULAR_GROUP ADC regular conversion type. – ADC_INJECTED_GROUP ADC injected conversion type. – ADC_REGULAR_INJECTED_GROUP ADC regular and injected conversion type.
Return values	• **HAL:** status.

ADC_Enable

Function name	**HAL_StatusTypeDef ADC_Enable (ADC_HandleTypeDef * hadc)**
Function description	Enable the selected ADC.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.
Notes	• Prerequisite condition to use this function: ADC must be disabled and voltage regulator must be enabled (done into HAL_ADC_Init()).

ADC_Disable

Function name	**HAL_StatusTypeDef ADC_Disable (ADC_HandleTypeDef * hadc)**
Function description	Disable the selected ADC.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.
Notes	• Prerequisite condition to use this function: ADC conversions must be stopped.

ADC_DMAConvCplt

Function name	**void ADC_DMAConvCplt (DMA_HandleTypeDef * hdma)**

Function description	DMA transfer complete callback.
Parameters	• **hdma:** pointer to DMA handle.
Return values	• **None:**

ADC_DMAHalfConvCplt

Function name	**void ADC_DMAHalfConvCplt (DMA_HandleTypeDef * hdma)**
Function description	DMA half transfer complete callback.
Parameters	• **hdma:** pointer to DMA handle.
Return values	• **None:**

ADC_DMAError

Function name	**void ADC_DMAError (DMA_HandleTypeDef * hdma)**
Function description	DMA error callback.
Parameters	• **hdma:** pointer to DMA handle.
Return values	• **None:**

6.3 ADC Firmware driver defines

6.3.1 ADC

ADC Analog Watchdog Mode

ADC_ANALOGWATCHDOG_NONE	No analog watchdog selected
ADC_ANALOGWATCHDOG_SINGLE_REG	Analog watchdog applied to a regular group single channel
ADC_ANALOGWATCHDOG_SINGLE_INJEC	Analog watchdog applied to an injected group single channel
ADC_ANALOGWATCHDOG_SINGLE_REGINJEC	Analog watchdog applied to a regular and injected groups single channel
ADC_ANALOGWATCHDOG_ALL_REG	Analog watchdog applied to regular group all channels
ADC_ANALOGWATCHDOG_ALL_INJEC	Analog watchdog applied to injected group all channels
ADC_ANALOGWATCHDOG_ALL_REGINJEC	Analog watchdog applied to regular and injected groups all channels

ADC Analog Watchdog Selection

ADC_ANALOGWATCHDOG_1	Analog watchdog 1 selection
ADC_ANALOGWATCHDOG_2	Analog watchdog 2 selection
ADC_ANALOGWATCHDOG_3	Analog watchdog 3 selection

ADCx CFGR fields

ADC_CFGR_FIELDS

ADCx CFGR sub fields

ADC_CFGR_FIELDS_2

ADC channels

ADC_CHANNEL_0	ADC channel 0
ADC_CHANNEL_1	ADC channel 1
ADC_CHANNEL_2	ADC channel 2
ADC_CHANNEL_3	ADC channel 3
ADC_CHANNEL_4	ADC channel 4
ADC_CHANNEL_5	ADC channel 5
ADC_CHANNEL_6	ADC channel 6
ADC_CHANNEL_7	ADC channel 7
ADC_CHANNEL_8	ADC channel 8
ADC_CHANNEL_9	ADC channel 9
ADC_CHANNEL_10	ADC channel 10
ADC_CHANNEL_11	ADC channel 11
ADC_CHANNEL_12	ADC channel 12
ADC_CHANNEL_13	ADC channel 13
ADC_CHANNEL_14	ADC channel 14
ADC_CHANNEL_15	ADC channel 15
ADC_CHANNEL_16	ADC channel 16
ADC_CHANNEL_17	ADC channel 17
ADC_CHANNEL_18	ADC channel 18
ADC_CHANNEL_TEMPSENSOR	ADC temperature sensor channel
ADC_CHANNEL_VBAT	ADC Vbat channel
ADC_CHANNEL_VREFINT	ADC Vrefint channel
ADC_CHANNEL_DAC1CH1	ADC internal channel connected to DAC1 channel 1, channel specific to ADC1. This channel is shared with ADC internal channel connected to temperature sensor, they cannot be used both simultenaeously.
ADC_CHANNEL_DAC1CH2	ADC internal channel connected to DAC1 channel 2, channel specific to ADC1. This channel is shared with ADC internal channel connected to Vbat, they cannot be used both simultenaeously.

ADC clock source and clock prescaler

ADC_CLOCK_SYNC_PCLK_DIV1	ADC synchronous clock derived from AHB clock not divided
ADC_CLOCK_SYNC_PCLK_DIV2	ADC synchronous clock derived from AHB clock divided a prescaler of 2
ADC_CLOCK_SYNC_PCLK_DIV4	ADC synchronous clock derived from AHB

	clock divided a prescaler of 4
ADC_CLOCKPRESCALER_PCLK_DIV1	Obsolete naming, kept for compatibility with some other devices
ADC_CLOCKPRESCALER_PCLK_DIV2	Obsolete naming, kept for compatibility with some other devices
ADC_CLOCKPRESCALER_PCLK_DIV4	Obsolete naming, kept for compatibility with some other devices
ADC_CLOCK_ASYNC_DIV1	ADC asynchronous clock not divided
ADC_CLOCK_ASYNC_DIV2	ADC asynchronous clock divided by 2
ADC_CLOCK_ASYNC_DIV4	ADC asynchronous clock divided by 4
ADC_CLOCK_ASYNC_DIV6	ADC asynchronous clock divided by 6
ADC_CLOCK_ASYNC_DIV8	ADC asynchronous clock divided by 8
ADC_CLOCK_ASYNC_DIV10	ADC asynchronous clock divided by 10
ADC_CLOCK_ASYNC_DIV12	ADC asynchronous clock divided by 12
ADC_CLOCK_ASYNC_DIV16	ADC asynchronous clock divided by 16
ADC_CLOCK_ASYNC_DIV32	ADC asynchronous clock divided by 32
ADC_CLOCK_ASYNC_DIV64	ADC asynchronous clock divided by 64
ADC_CLOCK_ASYNC_DIV128	ADC asynchronous clock divided by 128
ADC_CLOCK_ASYNC_DIV256	ADC asynchronous clock divided by 256

ADC conversion data alignment

ADC_DATAALIGN_RIGHT	Data right alignment
ADC_DATAALIGN_LEFT	Data left alignment

ADC sequencer end of unitary conversion or sequence conversions

ADC_EOC_SINGLE_CONV	End of unitary conversion flag
ADC_EOC_SEQ_CONV	End of sequence conversions flag

ADC Error Code

HAL_ADC_ERROR_NONE	No error
HAL_ADC_ERROR_INTERNAL	ADC IP internal error (problem of clocking, enable/disable, erroneous state, ...)
HAL_ADC_ERROR_OVR	Overrun error
HAL_ADC_ERROR_DMA	DMA transfer error
HAL_ADC_ERROR_JQOVF	Injected context queue overflow error

ADC Event Type

ADC_EOSMP_EVENT	ADC End of Sampling event
ADC_AWD1_EVENT	ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 series)
ADC_AWD2_EVENT	ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 series)

ADC_AWD3_EVENT	ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 series)
ADC_OVR_EVENT	ADC overrun event
ADC_JQOVF_EVENT	ADC Injected Context Queue Overflow event

ADC Exported Constants

ADC_AWD_EVENT	ADC Analog watchdog 1 event: Naming for compatibility with other STM32 devices having only one analog watchdog

ADC Flags Definition

ADC_FLAG_RDY	ADC Ready flag
ADC_FLAG_EOSMP	ADC End of Sampling flag
ADC_FLAG_EOC	ADC End of Regular Conversion flag
ADC_FLAG_EOS	ADC End of Regular sequence of Conversions flag
ADC_FLAG_OVR	ADC overrun flag
ADC_FLAG_JEOC	ADC End of Injected Conversion flag
ADC_FLAG_JEOS	ADC End of Injected sequence of Conversions flag
ADC_FLAG_AWD1	ADC Analog watchdog 1 flag (main analog watchdog)
ADC_FLAG_AWD2	ADC Analog watchdog 2 flag (additional analog watchdog)
ADC_FLAG_AWD3	ADC Analog watchdog 3 flag (additional analog watchdog)
ADC_FLAG_JQOVF	ADC Injected Context Queue Overflow flag
ADC_FLAG_AWD	ADC Analog watchdog 1 flag: Naming for compatibility with other STM32 devices having only one analog watchdog
ADC_FLAG_ALL	ADC all flags
ADC_FLAG_POSTCONV_ALL	ADC post-conversion all flags

Channel - Sampling time

ADC_SAMPLETIME_2CYCLES_5	Sampling time 2.5 ADC clock cycles
ADC_SAMPLETIME_3CYCLES_5	Sampling time 3.5 ADC clock cycles. If selected, this sampling time replaces all sampling time 2.5 ADC clock cycles. These 2 sampling times cannot be used simultaneously.
ADC_SAMPLETIME_6CYCLES_5	Sampling time 6.5 ADC clock cycles
ADC_SAMPLETIME_12CYCLES_5	Sampling time 12.5 ADC clock cycles
ADC_SAMPLETIME_24CYCLES_5	Sampling time 24.5 ADC clock cycles
ADC_SAMPLETIME_47CYCLES_5	Sampling time 47.5 ADC clock cycles
ADC_SAMPLETIME_92CYCLES_5	Sampling time 92.5 ADC clock cycles
ADC_SAMPLETIME_247CYCLES_5	Sampling time 247.5 ADC clock cycles
ADC_SAMPLETIME_640CYCLES_5	Sampling time 640.5 ADC clock cycles

HAL ADC macro to manage HAL ADC handle, IT and flags.

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_ENABLE_IT

Description:

- Enable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  - ADC_IT_RDY, ADC Ready (ADRDY) interrupt source
  - ADC_IT_EOSMP, ADC End of Sampling interrupt source
  - ADC_IT_EOC, ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS, ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR, ADC overrun interrupt source
  - ADC_IT_JEOC, ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS, ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1, ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2, ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3, ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_JQOVF, ADC Injected Context Queue Overflow interrupt source.

Return value:

- None

__HAL_ADC_DISABLE_IT

Description:

- Disable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  – ADC_IT_RDY, ADC Ready (ADRDY) interrupt source
  – ADC_IT_EOSMP, ADC End of Sampling interrupt source
  – ADC_IT_EOC, ADC End of Regular Conversion interrupt source
  – ADC_IT_EOS, ADC End of Regular sequence of Conversions interrupt source
  – ADC_IT_OVR, ADC overrun interrupt source
  – ADC_IT_JEOC, ADC End of Injected Conversion interrupt source
  – ADC_IT_JEOS, ADC End of Injected sequence of Conversions interrupt source
  – ADC_IT_AWD1, ADC Analog watchdog 1 interrupt source (main analog watchdog)
  – ADC_IT_AWD2, ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  – ADC_IT_AWD3, ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  – ADC_IT_JQOVF, ADC Injected Context Queue Overflow interrupt source.

Return value:

- None

__HAL_ADC_GET_IT_SOURCE

Description:

- Checks if the specified ADC interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be one of the following values:
  – ADC_IT_RDY, ADC Ready (ADRDY) interrupt source
  – ADC_IT_EOSMP, ADC End of Sampling interrupt source
  – ADC_IT_EOC, ADC End of Regular Conversion interrupt source
  – ADC_IT_EOS, ADC End of Regular sequence of Conversions interrupt

source
- – ADC_IT_OVR, ADC overrun interrupt source
- – ADC_IT_JEOC, ADC End of Injected Conversion interrupt source
- – ADC_IT_JEOS, ADC End of Injected sequence of Conversions interrupt source
- – ADC_IT_AWD1, ADC Analog watchdog 1 interrupt source (main analog watchdog)
- – ADC_IT_AWD2, ADC Analog watchdog 2 interrupt source (additional analog watchdog)
- – ADC_IT_AWD3, ADC Analog watchdog 3 interrupt source (additional analog watchdog)
- – ADC_IT_JQOVF, ADC Injected Context Queue Overflow interrupt source.

Return value:

- State: of interruption (SET or RESET)

__HAL_ADC_GET_FLAG

Description:

- Check whether the specified ADC flag is set or not.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - – ADC_FLAG_RDY, ADC Ready (ADRDY) flag
  - – ADC_FLAG_EOSMP, ADC End of Sampling flag
  - – ADC_FLAG_EOC, ADC End of Regular Conversion flag
  - – ADC_FLAG_EOS, ADC End of Regular sequence of Conversions flag
  - – ADC_FLAG_OVR, ADC overrun flag
  - – ADC_FLAG_JEOC, ADC End of Injected Conversion flag
  - – ADC_FLAG_JEOS, ADC End of Injected sequence of Conversions flag
  - – ADC_FLAG_AWD1, ADC Analog watchdog 1 flag (main analog watchdog)
  - – ADC_FLAG_AWD2, ADC Analog watchdog 2 flag (additional analog watchdog)
  - – ADC_FLAG_AWD3, ADC Analog watchdog 3 flag (additional analog watchdog)

- ADC_FLAG_JQOVF, ADC Injected Context Queue Overflow flag.

Return value:

- State: of flag (TRUE or FALSE).

__HAL_ADC_CLEAR_FLAG

Description:

- Clear the specified ADC flag.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY, ADC Ready (ADRDY) flag
  - ADC_FLAG_EOSMP, ADC End of Sampling flag
  - ADC_FLAG_EOC, ADC End of Regular Conversion flag
  - ADC_FLAG_EOS, ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR, ADC overrun flag
  - ADC_FLAG_JEOC, ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS, ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1, ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2, ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3, ADC Analog watchdog 3 flag (additional analog watchdog)
  - ADC_FLAG_JQOVF, ADC Injected Context Queue Overflow flag.

Return value:

- None

Notes:

- Bit cleared bit by writing 1 (writing 0 has no effect on any bit of register ISR).

HAL ADC helper macro

__HAL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be

one of the following values:

- – ADC_CHANNEL_0
- – ADC_CHANNEL_1 (7)
- – ADC_CHANNEL_2 (7)
- – ADC_CHANNEL_3 (7)
- – ADC_CHANNEL_4 (7)
- – ADC_CHANNEL_5 (7)
- – ADC_CHANNEL_6
- – ADC_CHANNEL_7
- – ADC_CHANNEL_8
- – ADC_CHANNEL_9
- – ADC_CHANNEL_10
- – ADC_CHANNEL_11
- – ADC_CHANNEL_12
- – ADC_CHANNEL_13
- – ADC_CHANNEL_14
- – ADC_CHANNEL_15
- – ADC_CHANNEL_16
- – ADC_CHANNEL_17
- – ADC_CHANNEL_18
- – ADC_CHANNEL_VREFINT (1)
- – ADC_CHANNEL_TEMPSENSOR (4)
- – ADC_CHANNEL_VBAT (4)
- – ADC_CHANNEL_DAC1CH1 (5)
- – ADC_CHANNEL_DAC1CH2 (5)
- – ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
- – ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
- – ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
- – ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __HAL_ADC_CHANNEL_TO_DECIMAL_NB(ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__HAL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1 (7)
  - ADC_CHANNEL_2 (7)
  - ADC_CHANNEL_3 (7)
  - ADC_CHANNEL_4 (7)
  - ADC_CHANNEL_5 (7)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (4)
  - ADC_CHANNEL_VBAT (4)
  - ADC_CHANNEL_DAC1CH1 (5)
  - ADC_CHANNEL_DAC1CH2 (5)
  - ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  - ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

Notes:

- Example: __HAL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "ADC_CHANNEL_4".

__HAL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:

- ADC_CHANNEL_0
- ADC_CHANNEL_1 (7)
- ADC_CHANNEL_2 (7)
- ADC_CHANNEL_3 (7)
- ADC_CHANNEL_4 (7)
- ADC_CHANNEL_5 (7)
- ADC_CHANNEL_6
- ADC_CHANNEL_7
- ADC_CHANNEL_8
- ADC_CHANNEL_9
- ADC_CHANNEL_10
- ADC_CHANNEL_11
- ADC_CHANNEL_12
- ADC_CHANNEL_13
- ADC_CHANNEL_14
- ADC_CHANNEL_15
- ADC_CHANNEL_16
- ADC_CHANNEL_17
- ADC_CHANNEL_18
- ADC_CHANNEL_VREFINT (1)
- ADC_CHANNEL_TEMPSENSOR (4)
- ADC_CHANNEL_VBAT (4)
- ADC_CHANNEL_DAC1CH1 (5)
- ADC_CHANNEL_DAC1CH2 (5)
- ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
- ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
- ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
- ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): ADC_CHANNEL_1, ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1 (7)
  - ADC_CHANNEL_2 (7)
  - ADC_CHANNEL_3 (7)
  - ADC_CHANNEL_4 (7)
  - ADC_CHANNEL_5 (7)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (4)
  - ADC_CHANNEL_VBAT (4)
  - ADC_CHANNEL_DAC1CH1 (5)
  - ADC_CHANNEL_DAC1CH2 (5)
  - ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  - ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__HAL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (4)
  - ADC_CHANNEL_VBAT (4)
  - ADC_CHANNEL_DAC1CH1 (5)
  - ADC_CHANNEL_DAC1CH2 (5)
  - ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - ADC_CHANNEL_DAC1CH1_ADC3

(3)(6)
- ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__HAL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__HAL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__HAL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of to the data to be converted

This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__HAL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__HAL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

Parameters:

- __VREFINT_ADC_DATA__: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  – ADC_RESOLUTION_12B
  – ADC_RESOLUTION_10B
  – ADC_RESOLUTION_8B
  – ADC_RESOLUTION_6B

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 serie, calibration data of internal voltage reference VrefInt corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__HAL_ADC_CALC_TEMPERATURE

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  – ADC_RESOLUTION_12B
  – ADC_RESOLUTION_10B
  – ADC_RESOLUTION_8B
  – ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS_ADC_DATA - TS_CAL1) * (TS_CAL2_TEMP - TS_CAL1_TEMP)) / (TS_CAL2 - TS_CAL1) + TS_CAL1_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC Avg_Slope = (TS_CAL2 - TS_CAL1) / (TS_CAL2_TEMP - TS_CAL1_TEMP) TS_CAL1 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) TS_CAL2 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). On this STM32 serie, calibration data of temperature sensor corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__HAL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32L4, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__:

Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On STM32L4, refer to device datasheet parameter "V30" (corresponding to TS_CAL1).

- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). ADC measurement data must correspond to a resolution of 12bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 12 bits.

ADC group injected trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGINJECCONV_EDGE_NONE	Injected conversions hardware trigger detection disabled
ADC_EXTERNALTRIGINJECCONV_EDGE_RISING	Injected conversions hardware trigger detection on the rising edge
ADC_EXTERNALTRIGINJECCONV_EDGE_FALLING	Injected conversions hardware trigger detection on the falling edge
ADC_EXTERNALTRIGINJECCONV_EDGE_RISINGFALLING	Injected conversions hardware trigger detection on both the rising and falling edges

ADC group injected trigger source

ADC_EXTERNALTRIGINJEC_T1_TRGO	Event 0 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T1_CC4	Event 1 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T2_TRGO	Event 2 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T2_CC1	Event 3 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T3_CC4	Event 4 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T4_TRGO	Event 5 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_EXT_IT15	Event 6 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T8_CC4	Event 7 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T1_TRGO2	Event 8 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T8_TRGO	Event 9 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T8_TRGO2	Event 10 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T3_CC3	Event 11 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T3_TRGO	Event 12 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T3_CC1	Event 13 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T6_TRGO	Event 14 triggers injected group conversion start
ADC_EXTERNALTRIGINJEC_T15_TRGO	Event 15 triggers injected group conversion start
ADC_INJECTED_SOFTWARE_START	Software triggers injected group conversion start

ADC Interrupts Definition

ADC_IT_RDY	ADC Ready (ADRDY) interrupt source
ADC_IT_EOSMP	ADC End of sampling interrupt source
ADC_IT_EOC	ADC End of regular conversion interrupt source
ADC_IT_EOS	ADC End of regular sequence of conversions interrupt source
ADC_IT_OVR	ADC overrun interrupt source
ADC_IT_JEOC	ADC End of injected conversion interrupt source
ADC_IT_JEOS	ADC End of injected sequence of conversions interrupt source
ADC_IT_AWD1	ADC Analog watchdog 1 interrupt source (main analog watchdog)
ADC_IT_AWD2	ADC Analog watchdog 2 interrupt source (additional analog watchdog)
ADC_IT_AWD3	ADC Analog watchdog 3 interrupt source (additional analog watchdog)
ADC_IT_JQOVF	ADC Injected Context Queue Overflow interrupt source
ADC_IT_AWD	ADC Analog watchdog 1 interrupt source: naming for compatibility with other STM32 devices having only one analog watchdog

ADC overrun

ADC_OVR_DATA_PRESERVED	Data preserved in case of overrun
ADC_OVR_DATA_OVERWRITTEN	Data overwritten in case of overrun

ADC Oversampling Ratio

ADC_OVERSAMPLING_RATIO_2	ADC Oversampling ratio 2x
ADC_OVERSAMPLING_RATIO_4	ADC Oversampling ratio 4x
ADC_OVERSAMPLING_RATIO_8	ADC Oversampling ratio 8x
ADC_OVERSAMPLING_RATIO_16	ADC Oversampling ratio 16x
ADC_OVERSAMPLING_RATIO_32	ADC Oversampling ratio 32x
ADC_OVERSAMPLING_RATIO_64	ADC Oversampling ratio 64x

ADC_OVERSAMPLING_RATIO_128	ADC Oversampling ratio 128x
ADC_OVERSAMPLING_RATIO_256	ADC Oversampling ratio 256x

ADC group regular trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGCONVEDGE_NONE	Regular conversions hardware trigger detection disabled
ADC_EXTERNALTRIGCONVEDGE_RISING	Regular conversions hardware trigger detection on the rising edge
ADC_EXTERNALTRIGCONVEDGE_FALLING	Regular conversions hardware trigger detection on the falling edge
ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING	Regular conversions hardware trigger detection on both the rising and falling edges

ADC group regular trigger source

ADC_EXTERNALTRIG_T1_CC1	Event 0 triggers regular group conversion start
ADC_EXTERNALTRIG_T1_CC2	Event 1 triggers regular group conversion start
ADC_EXTERNALTRIG_T1_CC3	Event 2 triggers regular group conversion start
ADC_EXTERNALTRIG_T2_CC2	Event 3 triggers regular group conversion start
ADC_EXTERNALTRIG_T3_TRGO	Event 4 triggers regular group conversion start
ADC_EXTERNALTRIG_T4_CC4	Event 5 triggers regular group conversion start
ADC_EXTERNALTRIG_EXT_IT11	Event 6 triggers regular group conversion start
ADC_EXTERNALTRIG_T8_TRGO	Event 7 triggers regular group conversion start
ADC_EXTERNALTRIG_T8_TRGO2	Event 8 triggers regular group conversion start
ADC_EXTERNALTRIG_T1_TRGO	Event 9 triggers regular group conversion start
ADC_EXTERNALTRIG_T1_TRGO2	Event 10 triggers regular group conversion start
ADC_EXTERNALTRIG_T2_TRGO	Event 11 triggers regular group conversion start
ADC_EXTERNALTRIG_T4_TRGO	Event 12 triggers regular group conversion start
ADC_EXTERNALTRIG_T6_TRGO	Event 13 triggers regular group conversion start
ADC_EXTERNALTRIG_T15_TRGO	Event 14 triggers regular group conversion start
ADC_EXTERNALTRIG_T3_CC4	Event 15 triggers regular group conversion start
ADC_SOFTWARE_START	Software triggers regular group conversion start

ADC Regular Oversampling Continued or Resumed Mode

ADC_REGOVERSAMPLING_CONTINUED_MODE	Oversampling buffer maintained during injection sequence
ADC_REGOVERSAMPLING_RESUMED_MODE	Oversampling buffer zeroed during injection sequence

ADC group regular sequencer rank

ADC_REGULAR_RANK_1	ADC regular conversion rank 1

ADC_REGULAR_RANK_2	ADC regular conversion rank 2
ADC_REGULAR_RANK_3	ADC regular conversion rank 3
ADC_REGULAR_RANK_4	ADC regular conversion rank 4
ADC_REGULAR_RANK_5	ADC regular conversion rank 5
ADC_REGULAR_RANK_6	ADC regular conversion rank 6
ADC_REGULAR_RANK_7	ADC regular conversion rank 7
ADC_REGULAR_RANK_8	ADC regular conversion rank 8
ADC_REGULAR_RANK_9	ADC regular conversion rank 9
ADC_REGULAR_RANK_10	ADC regular conversion rank 10
ADC_REGULAR_RANK_11	ADC regular conversion rank 11
ADC_REGULAR_RANK_12	ADC regular conversion rank 12
ADC_REGULAR_RANK_13	ADC regular conversion rank 13
ADC_REGULAR_RANK_14	ADC regular conversion rank 14
ADC_REGULAR_RANK_15	ADC regular conversion rank 15
ADC_REGULAR_RANK_16	ADC regular conversion rank 16

ADC Resolution

ADC_RESOLUTION_12B	ADC 12-bit resolution
ADC_RESOLUTION_10B	ADC 10-bit resolution
ADC_RESOLUTION_8B	ADC 8-bit resolution
ADC_RESOLUTION_6B	ADC 6-bit resolution

ADC Oversampling Right Shift

ADC_RIGHTBITSHIFT_NONE	ADC No bit shift for oversampling
ADC_RIGHTBITSHIFT_1	ADC 1 bit shift for oversampling
ADC_RIGHTBITSHIFT_2	ADC 2 bits shift for oversampling
ADC_RIGHTBITSHIFT_3	ADC 3 bits shift for oversampling
ADC_RIGHTBITSHIFT_4	ADC 4 bits shift for oversampling
ADC_RIGHTBITSHIFT_5	ADC 5 bits shift for oversampling
ADC_RIGHTBITSHIFT_6	ADC 6 bits shift for oversampling
ADC_RIGHTBITSHIFT_7	ADC 7 bits shift for oversampling
ADC_RIGHTBITSHIFT_8	ADC 8 bits shift for oversampling

ADC sequencer scan mode

ADC_SCAN_DISABLE	Scan mode disabled
ADC_SCAN_ENABLE	Scan mode enabled

ADCx SMPR1 fields

ADC_SMPR1_FIELDS

ADC States

HAL_ADC_STATE_RESET	**Notes:** • ADC state machine is managed by bitfields, state must be compared with bit by bit. For example: " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_REG_BUSY)) " " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_AWD1)) " ADC not yet initialized or disabled
HAL_ADC_STATE_READY	ADC peripheral ready for use
HAL_ADC_STATE_BUSY_INTERNAL	ADC is busy due to an internal process (initialization, calibration)
HAL_ADC_STATE_TIMEOUT	TimeOut occurrence
HAL_ADC_STATE_ERROR_INTERNAL	Internal error occurrence
HAL_ADC_STATE_ERROR_CONFIG	Configuration error occurrence
HAL_ADC_STATE_ERROR_DMA	DMA error occurrence
HAL_ADC_STATE_REG_BUSY	A conversion on ADC group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))
HAL_ADC_STATE_REG_EOC	Conversion data available on group regular
HAL_ADC_STATE_REG_OVR	Overrun occurrence
HAL_ADC_STATE_REG_EOSMP	Not available on this STM32 serie: End Of Sampling flag raised
HAL_ADC_STATE_INJ_BUSY	A conversion on ADC group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))
HAL_ADC_STATE_INJ_EOC	Conversion data available on group injected
HAL_ADC_STATE_INJ_JQOVF	Injected queue overflow occurrence
HAL_ADC_STATE_AWD1	Out-of-window occurrence of ADC analog watchdog 1
HAL_ADC_STATE_AWD2	Out-of-window occurrence of ADC analog watchdog 2
HAL_ADC_STATE_AWD3	Out-of-window occurrence of ADC analog watchdog 3
HAL_ADC_STATE_MULTIMODE_SLAVE	ADC in multimode slave state, controlled by another ADC master (when feature available)

ADC Triggered Regular Oversampling

ADC_TRIGGEREDMODE_SINGLE_TRIGGER	A single trigger for all channel oversampled conversions
ADC_TRIGGEREDMODE_MULTI_TRIGGER	A trigger for each oversampled conversion

7 HAL ADC Extension Driver

7.1 ADCEx Firmware driver registers structures

7.1.1 ADC_InjOversamplingTypeDef

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***

Field Documentation

- ***uint32_t ADC_InjOversamplingTypeDef::Ratio***
  Configures the oversampling ratio. This parameter can be a value of ***ADC_Oversampling_Ratio***
- ***uint32_t ADC_InjOversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_Right_Bit_Shift***

7.1.2 ADC_InjectionConfTypeDef

Data Fields

- ***uint32_t InjectedChannel***
- ***uint32_t InjectedRank***
- ***uint32_t InjectedSamplingTime***
- ***uint32_t InjectedSingleDiff***
- ***uint32_t InjectedOffsetNumber***
- ***uint32_t InjectedOffset***
- ***uint32_t InjectedNbrOfConversion***
- ***uint32_t InjectedDiscontinuousConvMode***
- ***uint32_t AutoInjectedConv***
- ***uint32_t QueueInjectedContext***
- ***uint32_t ExternalTrigInjecConv***
- ***uint32_t ExternalTrigInjecConvEdge***
- ***uint32_t InjecOversamplingMode***
- ***ADC_InjOversamplingTypeDef InjecOversampling***

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel***
  Specifies the channel to configure into ADC group injected. This parameter can be a value of ***ADC_channels*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank***
  Specifies the rank in the ADC group injected sequencer. This parameter must be a value of ***ADCEx_injected_rank***. Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)
- ***uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime***
  Sampling time value to be set for the selected channel. Unit: ADC clock cycles. Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles

at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values.

- ***uint32_t ADC_InjectionConfTypeDef::InjectedSingleDiff***
  Selection of single-ended or differential input. In differential mode: Differential measurement is between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of ***ADCEx_SingleDifferential***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetNumber***
  Selects the offset number. This parameter can be a value of ***ADCEx_OffsetNumber***. Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset***
  Defines the offset to be subtracted from the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion***
  Specifies the number of ranks that will be converted within the ADC group injected sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode***
  Specifies whether the conversions sequence of ADC group injected is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). Note: For injected group, discontinuous mode converts the sequence channel by channel (discontinuous length fixed to 1 rank). Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::AutoInjectedConv***
  Enables or disables the selected ADC group injected automatic conversion after regular one This parameter can be set to ENABLE or DISABLE. Note: To use

Automatic injected conversion, discontinuous mode must be disabled ('DiscontinuousConvMode' and 'InjectedDiscontinuousConvMode' set to DISABLE) Note: To use Automatic injected conversion, injected group external triggers must be disabled ('ExternalTrigInjecConv' set to ADC_INJECTED_SOFTWARE_START) Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***uint32_t ADC_InjectionConfTypeDef::QueueInjectedContext***
  Specifies whether the context queue feature is enabled. This parameter can be set to ENABLE or DISABLE. If context queue is enabled, injected sequencer&channels configurations are queued on up to 2 contexts. If a new injected context is set when queue is full, error is triggered by interruption and through function 'HAL_ADCEx_InjectedQueueOverflowCallback'. Caution: This feature request that the sequence is fully configured before injected conversion start. Therefore, configure channels with as many calls to **HAL_ADCEx_InjectedConfigChannel()** as the 'InjectedNbrOfConversion' parameter. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion).
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***
  Selects the external event used to trigger the conversion start of injected group. If set to ADC_INJECTED_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_injected_external_trigger_source***. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge***
  Selects the external trigger edge of injected group. This parameter can be a value of ***ADC_injected_external_trigger_edge***. If trigger source is set to ADC_INJECTED_SOFTWARE_START, this parameter is discarded. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::InjecOversamplingMode***
  Specifies whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).
- ***ADC_InjOversamplingTypeDef ADC_InjectionConfTypeDef::InjecOversampling***
  Specifies the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling already enabled. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).

7.2 ADCEx Firmware driver API description

7.2.1 IO operation functions

This section provides functions allowing to:

- Perform the ADC self-calibration for single or differential ending.
- Get calibration factors for single or differential ending.

- Set calibration factors for single or differential ending.
- Start conversion of ADC group injected.
- Stop conversion of ADC group injected.
- Poll for conversion complete on ADC group injected.
- Get result of ADC group injected channel conversion.
- Start conversion of ADC group injected and enable interruptions.
- Stop conversion of ADC group injected and disable interruptions.
- When multimode feature is available, start multimode and enable DMA transfer.
- Stop multimode and disable ADC DMA transfer.
- Get result of multimode conversion.

This section contains the following APIs:

- *HAL_ADCEx_Calibration_Start()*
- *HAL_ADCEx_Calibration_GetValue()*
- *HAL_ADCEx_Calibration_SetValue()*
- *HAL_ADCEx_InjectedStart()*
- *HAL_ADCEx_InjectedStop()*
- *HAL_ADCEx_InjectedPollForConversion()*
- *HAL_ADCEx_InjectedStart_IT()*
- *HAL_ADCEx_InjectedStop_IT()*
- *HAL_ADCEx_InjectedGetValue()*
- *HAL_ADCEx_InjectedConvCpltCallback()*
- *HAL_ADCEx_InjectedQueueOverflowCallback()*
- *HAL_ADCEx_LevelOutOfWindow2Callback()*
- *HAL_ADCEx_LevelOutOfWindow3Callback()*
- *HAL_ADCEx_EndOfSamplingCallback()*
- *HAL_ADCEx_RegularStop()*
- *HAL_ADCEx_RegularStop_IT()*
- *HAL_ADCEx_RegularStop_DMA()*

7.2.2 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on injected group
- Configure multimode when multimode feature is available
- Enable or Disable Injected Queue
- Disable ADC voltage regulator
- Enter ADC deep-power-down mode

This section contains the following APIs:

- *HAL_ADCEx_InjectedConfigChannel()*
- *HAL_ADCEx_EnableInjectedQueue()*
- *HAL_ADCEx_DisableInjectedQueue()*
- *HAL_ADCEx_DisableVoltageRegulator()*
- *HAL_ADCEx_EnterADCDeepPowerDownMode()*

7.2.3 Detailed description of functions

HAL_ADCEx_Calibration_Start

Function name	**HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)**

Function description	Perform an ADC automatic self-calibration Calibration prerequisite: ADC must be disabled (execute this function before HAL_ADC_Start() or after HAL_ADC_Stop()).
Parameters	• **hadc:** ADC handle • **SingleDiff:** Selection of single-ended or differential input This parameter can be one of the following values: – ADC_SINGLE_ENDED Channel in mode input single ended – ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended
Return values	• **HAL:** status

HAL_ADCEx_Calibration_GetValue

Function name	**uint32_t HAL_ADCEx_Calibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)**
Function description	Get the calibration factor.
Parameters	• **hadc:** ADC handle. • **SingleDiff:** This parameter can be only: – ADC_SINGLE_ENDED Channel in mode input single ended – ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended
Return values	• **Calibration:** value.

HAL_ADCEx_Calibration_SetValue

Function name	**HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)**
Function description	Set the calibration factor to overwrite automatic conversion result.
Parameters	• **hadc:** ADC handle • **SingleDiff:** This parameter can be only: – ADC_SINGLE_ENDED Channel in mode input single ended – ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended • **CalibrationFactor:** Calibration factor (coded on 7 bits maximum)
Return values	• **HAL:** state

HAL_ADCEx_InjectedStart

Function name	**HAL_StatusTypeDef HAL_ADCEx_InjectedStart (ADC_HandleTypeDef * hadc)**
Function description	Enable ADC, start conversion of injected group.
Parameters	• **hadc:** ADC handle.
Return values	• **HAL:** status

Notes	• Interruptions enabled in this function: None. • Case of multimode enabled when multimode feature is available: HAL_ADCEx_InjectedStart() API must be called for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop

Function name	**HAL_StatusTypeDef HAL_ADCEx_InjectedStop (ADC_HandleTypeDef * hadc)**
Function description	Stop conversion of injected channels.
Parameters	• **hadc:** ADC handle.
Return values	• **HAL:** status
Notes	• If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC. • If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used. • In case of multimode enabled (when multimode feature is available), HAL_ADCEx_InjectedStop() must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).

HAL_ADCEx_InjectedPollForConversion

Function name	**HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)**
Function description	Wait for injected group conversion to be completed.
Parameters	• **hadc:** ADC handle • **Timeout:** Timeout value in millisecond.
Return values	• **HAL:** status
Notes	• Depending on hadc->Init.EOCSelection, JEOS or JEOC is checked and cleared depending on AUTDLY bit status.

HAL_ADCEx_InjectedStart_IT

Function name	**HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (ADC_HandleTypeDef * hadc)**
Function description	Enable ADC, start conversion of injected group with interruption.
Parameters	• **hadc:** ADC handle.
Return values	• **HAL:** status.
Notes	• Interruptions enabled in this function according to initialization setting: JEOC (end of conversion) or JEOS (end of sequence) • Case of multimode enabled (when multimode feature is enabled): HAL_ADCEx_InjectedStart_IT() API must be called

for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop_IT

Function name	**HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)**
Function description	Stop conversion of injected channels, disable interruption of end-of-conversion.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status
Notes	• If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC. • If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used. • Case of multimode enabled (when multimode feature is available): HAL_ADCEx_InjectedStop_IT() API must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave). • In case of auto-injection mode, HAL_ADC_Stop() must be used.

HAL_ADCEx_InjectedGetValue

Function name	**uint32_t HAL_ADCEx_InjectedGetValue (ADC_HandleTypeDef * hadc, uint32_t InjectedRank)**
Function description	Get ADC injected group conversion result.
Parameters	• **hadc:** ADC handle • **InjectedRank:** the converted ADC injected rank. This parameter can be one of the following values: – ADC_INJECTED_RANK_1 ADC group injected rank 1 – ADC_INJECTED_RANK_2 ADC group injected rank 2 – ADC_INJECTED_RANK_3 ADC group injected rank 3 – ADC_INJECTED_RANK_4 ADC group injected rank 4
Return values	• **ADC:** group injected conversion data
Notes	• Reading register JDRx automatically clears ADC flag JEOC (ADC group injected end of unitary conversion). • This function does not clear ADC flag JEOS (ADC group injected end of sequence conversion) Occurrence of flag JEOS rising: If sequencer is composed of 1 rank, flag JEOS is equivalent to flag JEOC.If sequencer is composed of several ranks, during the scan sequence flag JEOC only is raised, at the end of the scan sequence both flags JEOC and EOS are raised. Flag JEOS must not be cleared by this function because it would not be compliant with low power features (feature low power auto-wait, not available on all STM32

families). To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADCEx_InjectedPollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_JEOS).

HAL_ADCEx_InjectedConvCpltCallback

Function name	**void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)**
Function description	Injected conversion complete callback in non-blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADCEx_InjectedQueueOverflowCallback

Function name	**void HAL_ADCEx_InjectedQueueOverflowCallback (ADC_HandleTypeDef * hadc)**
Function description	Injected context queue overflow callback.
Parameters	• **hadc:** ADC handle
Return values	• **None:**
Notes	• This callback is called if injected context queue is enabled (parameter "QueueInjectedContext" in injected channel configuration) and if a new injected context is set when queue is full (maximum 2 contexts).

HAL_ADCEx_LevelOutOfWindow2Callback

Function name	**void HAL_ADCEx_LevelOutOfWindow2Callback (ADC_HandleTypeDef * hadc)**
Function description	Analog watchdog 2 callback in non-blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADCEx_LevelOutOfWindow3Callback

Function name	**void HAL_ADCEx_LevelOutOfWindow3Callback (ADC_HandleTypeDef * hadc)**
Function description	Analog watchdog 3 callback in non-blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADCEx_EndOfSamplingCallback

Function name	**void HAL_ADCEx_EndOfSamplingCallback (ADC_HandleTypeDef * hadc)**

Function description	End Of Sampling callback in non-blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADCEx_RegularStop

Function name	**HAL_StatusTypeDef HAL_ADCEx_RegularStop (ADC_HandleTypeDef * hadc)**
Function description	Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral if no conversion is on going on injected group.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.

HAL_ADCEx_RegularStop_IT

Function name	**HAL_StatusTypeDef HAL_ADCEx_RegularStop_IT (ADC_HandleTypeDef * hadc)**
Function description	Stop ADC conversion of ADC groups regular and injected, disable interrution of end-of-conversion, disable ADC peripheral if no conversion is on going on injected group.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.

HAL_ADCEx_RegularStop_DMA

Function name	**HAL_StatusTypeDef HAL_ADCEx_RegularStop_DMA (ADC_HandleTypeDef * hadc)**
Function description	Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral if no conversion is on going on injected group.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status.
Notes	• HAL_ADCEx_RegularStop_DMA() function is dedicated to single-ADC mode only. For multimode (when multimode feature is available), HAL_ADCEx_RegularMultiModeStop_DMA() API must be used.

HAL_ADCEx_InjectedConfigChannel

Function name	**HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef * sConfigInjected)**
Function description	Configure a channel to be assigned to ADC group injected.
Parameters	• **hadc:** ADC handle • **sConfigInjected:** Structure of ADC injected group and ADC

channel for injected group.

Return values
- **HAL:** status

Notes
- Possibility to update parameters on the fly: This function initializes injected group, following calls to this function can be used to reconfigure some parameters of structure "ADC_InjectionConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_InjectionConfTypeDef".
- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit().
- Caution: For Injected Context Queue use, a context must be fully defined before start of injected conversion. All channels are configured consecutively for the same ADC instance. Therefore, the number of calls to HAL_ADCEx_InjectedConfigChannel() must be equal to the value of parameter InjectedNbrOfConversion for each context. Example 1: If 1 context is intended to be used (or if there is no use of the Injected Queue Context feature) and if the context contains 3 injected ranks (InjectedNbrOfConversion = 3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel (i.e. 3 times) before starting a conversion. This function must not be called to configure a 4th injected channel: it would start a new context into context queue.Example 2: If 2 contexts are intended to be used and each of them contains 3 injected ranks (InjectedNbrOfConversion = 3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel and for each context (3 channels x 2 contexts = 6 calls). Conversion can start once the 1st context is set, that is after the first three HAL_ADCEx_InjectedConfigChannel() calls. The 2nd context can be set on the fly.

HAL_ADCEx_EnableInjectedQueue

Function name
HAL_StatusTypeDef HAL_ADCEx_EnableInjectedQueue (ADC_HandleTypeDef * hadc)

Function description
Enable Injected Queue.

Parameters
- **hadc:** ADC handle

Return values
- **HAL:** status

Notes
- This function resets CFGR register JQDIS bit in order to enable the Injected Queue. JQDIS can be written only when ADSTART and JDSTART are both equal to 0 to ensure that no regular nor injected conversion is ongoing.

HAL_ADCEx_DisableInjectedQueue

Function name
HAL_StatusTypeDef HAL_ADCEx_DisableInjectedQueue

	(ADC_HandleTypeDef * hadc)
Function description	Disable Injected Queue.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status
Notes	• This function sets CFGR register JQDIS bit in order to disable the Injected Queue. JQDIS can be written only when ADSTART and JDSTART are both equal to 0 to ensure that no regular nor injected conversion is ongoing.

HAL_ADCEx_DisableVoltageRegulator

Function name	**HAL_StatusTypeDef HAL_ADCEx_DisableVoltageRegulator (ADC_HandleTypeDef * hadc)**
Function description	Disable ADC voltage regulator.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status
Notes	• Disabling voltage regulator allows to save power. This operation can be carried out only when ADC is disabled. • To enable again the voltage regulator, the user is expected to resort to HAL_ADC_Init() API.

HAL_ADCEx_EnterADCDeepPowerDownMode

Function name	**HAL_StatusTypeDef HAL_ADCEx_EnterADCDeepPowerDownMode (ADC_HandleTypeDef * hadc)**
Function description	Enter ADC deep-power-down mode.
Parameters	• **hadc:** ADC handle
Return values	• **HAL:** status
Notes	• This mode is achieved in setting DEEPPWD bit and allows to save power in reducing leakage currents. It is particularly interesting before entering stop modes. • Setting DEEPPWD automatically clears ADVREGEN bit and disables the ADC voltage regulator. This means that this API encompasses HAL_ADCEx_DisableVoltageRegulator(). Additionally, the internal calibration is lost. • To exit the ADC deep-power-down mode, the user is expected to resort to HAL_ADC_Init() API as well as to relaunch a calibration with HAL_ADCEx_Calibration_Start() API or to re-apply a previously saved calibration factor.

7.3 ADCEx Firmware driver defines

7.3.1 ADCEx

ADC Extended Conversion Group

ADC_REGULAR_GROUP	ADC regular group selection

ADC_INJECTED_GROUP	ADC injected group selection
ADC_REGULAR_INJECTED_GROUP	ADC regular and injected groups selection

ADC Extended DFSDM mode configuration

ADC_DFSDM_MODE_DISABLE	ADC conversions are not transferred by DFSDM.
ADC_DFSDM_MODE_ENABLE	ADC conversion data are transfered to DFSDM for post processing. The ADC conversion data format must be 16-bit signed and right aligned, refer to reference manual. DFSDM transfer cannot be used if DMA transfer is enabled.

ADC Extended Injected Channel Rank

ADC_INJECTED_RANK_1	ADC injected conversion rank 1
ADC_INJECTED_RANK_1	ADC injected conversion rank 1
ADC_INJECTED_RANK_2	ADC injected conversion rank 2
ADC_INJECTED_RANK_2	ADC injected conversion rank 2
ADC_INJECTED_RANK_3	ADC injected conversion rank 3
ADC_INJECTED_RANK_3	ADC injected conversion rank 3
ADC_INJECTED_RANK_4	ADC injected conversion rank 4
ADC_INJECTED_RANK_4	ADC injected conversion rank 4

ADC Extended Offset Number

ADC_OFFSET_NONE	No offset correction
ADC_OFFSET_1	Offset correction to apply to a first channel
ADC_OFFSET_2	Offset correction to apply to a second channel
ADC_OFFSET_3	Offset correction to apply to a third channel
ADC_OFFSET_4	Offset correction to apply to a fourth channel

ADC Extended Single-ended/Differential input mode

ADC_SINGLE_ENDED	ADC channel set in single-ended input mode
ADC_DIFFERENTIAL_ENDED	ADC channel set in differential mode

8 HAL CAN Generic Driver

8.1 CAN Firmware driver registers structures

8.1.1 CAN_InitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Mode***
- ***uint32_t SJW***
- ***uint32_t BS1***
- ***uint32_t BS2***
- ***uint32_t TTCM***
- ***uint32_t ABOM***
- ***uint32_t AWUM***
- ***uint32_t NART***
- ***uint32_t RFLM***
- ***uint32_t TXFP***

Field Documentation

- ***uint32_t CAN_InitTypeDef::Prescaler***
  Specifies the length of a time quantum. This parameter must be a number between Min_Data = 1 and Max_Data = 1024
- ***uint32_t CAN_InitTypeDef::Mode***
  Specifies the CAN operating mode. This parameter can be a value of ***CAN_operating_mode***
- ***uint32_t CAN_InitTypeDef::SJW***
  Specifies the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter can be a value of ***CAN_synchronisation_jump_width***
- ***uint32_t CAN_InitTypeDef::BS1***
  Specifies the number of time quanta in Bit Segment 1. This parameter can be a value of ***CAN_time_quantum_in_bit_segment_1***
- ***uint32_t CAN_InitTypeDef::BS2***
  Specifies the number of time quanta in Bit Segment 2. This parameter can be a value of ***CAN_time_quantum_in_bit_segment_2***
- ***uint32_t CAN_InitTypeDef::TTCM***
  Enable or disable the time triggered communication mode. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t CAN_InitTypeDef::ABOM***
  Enable or disable the automatic bus-off management. This parameter can be set to ENABLE or DISABLE
- ***uint32_t CAN_InitTypeDef::AWUM***
  Enable or disable the automatic wake-up mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t CAN_InitTypeDef::NART***
  Enable or disable the non-automatic retransmission mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t CAN_InitTypeDef::RFLM***
  Enable or disable the receive FIFO Locked mode. This parameter can be set to ENABLE or DISABLE

- ***uint32_t CAN_InitTypeDef::TXFP***
  Enable or disable the transmit FIFO priority. This parameter can be set to ENABLE or DISABLE

8.1.2 CAN_FilterConfTypeDef

Data Fields

- ***uint32_t FilterIdHigh***
- ***uint32_t FilterIdLow***
- ***uint32_t FilterMaskIdHigh***
- ***uint32_t FilterMaskIdLow***
- ***uint32_t FilterFIFOAssignment***
- ***uint32_t FilterNumber***
- ***uint32_t FilterMode***
- ***uint32_t FilterScale***
- ***uint32_t FilterActivation***
- ***uint32_t BankNumber***

Field Documentation

- ***uint32_t CAN_FilterConfTypeDef::FilterIdHigh***
  Specifies the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t CAN_FilterConfTypeDef::FilterIdLow***
  Specifies the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t CAN_FilterConfTypeDef::FilterMaskIdHigh***
  Specifies the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t CAN_FilterConfTypeDef::FilterMaskIdLow***
  Specifies the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t CAN_FilterConfTypeDef::FilterFIFOAssignment***
  Specifies the FIFO (0 or 1) which will be assigned to the filter. This parameter can be a value of ***CAN_filter_FIFO***
- ***uint32_t CAN_FilterConfTypeDef::FilterNumber***
  Specifies the filter which will be initialized. This parameter must be a number between Min_Data = 0 and Max_Data = 27
- ***uint32_t CAN_FilterConfTypeDef::FilterMode***
  Specifies the filter mode to be initialized. This parameter can be a value of ***CAN_filter_mode***
- ***uint32_t CAN_FilterConfTypeDef::FilterScale***
  Specifies the filter scale. This parameter can be a value of ***CAN_filter_scale***
- ***uint32_t CAN_FilterConfTypeDef::FilterActivation***
  Enable or disable the filter. This parameter can be set to ENABLE or DISABLE
- ***uint32_t CAN_FilterConfTypeDef::BankNumber***
  Select the start slave bank filter. This parameter must be a number between Min_Data = 0 and Max_Data = 28

8.1.3 CanTxMsgTypeDef

Data Fields

- ***uint32_t StdId***
- ***uint32_t ExtId***
- ***uint32_t IDE***
- ***uint32_t RTR***
- ***uint32_t DLC***
- ***uint8_t Data***

Field Documentation

- ***uint32_t CanTxMsgTypeDef::StdId***
  Specifies the standard identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x7FF
- ***uint32_t CanTxMsgTypeDef::ExtId***
  Specifies the extended identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x1FFFFFFF
- ***uint32_t CanTxMsgTypeDef::IDE***
  Specifies the type of identifier for the message that will be transmitted. This parameter can be a value of ***CAN_identifier_type***
- ***uint32_t CanTxMsgTypeDef::RTR***
  Specifies the type of frame for the message that will be transmitted. This parameter can be a value of ***CAN_remote_transmission_request***
- ***uint32_t CanTxMsgTypeDef::DLC***
  Specifies the length of the frame that will be transmitted. This parameter must be a number between Min_Data = 0 and Max_Data = 8
- ***uint8_t CanTxMsgTypeDef::Data[8]***
  Contains the data to be transmitted. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFF

8.1.4 CanRxMsgTypeDef

Data Fields

- ***uint32_t StdId***
- ***uint32_t ExtId***
- ***uint32_t IDE***
- ***uint32_t RTR***
- ***uint32_t DLC***
- ***uint8_t Data***
- ***uint32_t FMI***
- ***uint32_t FIFONumber***

Field Documentation

- ***uint32_t CanRxMsgTypeDef::StdId***
  Specifies the standard identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x7FF
- ***uint32_t CanRxMsgTypeDef::ExtId***
  Specifies the extended identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x1FFFFFFF
- ***uint32_t CanRxMsgTypeDef::IDE***
  Specifies the type of identifier for the message that will be received. This parameter can be a value of ***CAN_identifier_type***

- ***uint32_t CanRxMsgTypeDef::RTR***
  Specifies the type of frame for the received message. This parameter can be a value of ***CAN_remote_transmission_request***
- ***uint32_t CanRxMsgTypeDef::DLC***
  Specifies the length of the frame that will be received. This parameter must be a number between Min_Data = 0 and Max_Data = 8
- ***uint8_t CanRxMsgTypeDef::Data[8]***
  Contains the data to be received. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFF
- ***uint32_t CanRxMsgTypeDef::FMI***
  Specifies the index of the filter the message stored in the mailbox passes through. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFF
- ***uint32_t CanRxMsgTypeDef::FIFONumber***
  Specifies the receive FIFO number. This parameter can be CAN_FIFO0 or CAN_FIFO1

8.1.5 CAN_HandleTypeDef

Data Fields

- ***CAN_TypeDef * Instance***
- ***CAN_InitTypeDef Init***
- ***CanTxMsgTypeDef * pTxMsg***
- ***CanRxMsgTypeDef * pRxMsg***
- ***__IO HAL_CAN_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***CAN_TypeDef* CAN_HandleTypeDef::Instance***
  Register base address
- ***CAN_InitTypeDef CAN_HandleTypeDef::Init***
  CAN required parameters
- ***CanTxMsgTypeDef* CAN_HandleTypeDef::pTxMsg***
  Pointer to transmit structure
- ***CanRxMsgTypeDef* CAN_HandleTypeDef::pRxMsg***
  Pointer to reception structure
- ***__IO HAL_CAN_StateTypeDef CAN_HandleTypeDef::State***
  CAN communication state
- ***HAL_LockTypeDef CAN_HandleTypeDef::Lock***
  CAN locking object
- ***__IO uint32_t CAN_HandleTypeDef::ErrorCode***
  CAN Error code

8.2 CAN Firmware driver API description

8.2.1 How to use this driver

1. Enable the CAN controller interface clock using __HAL_RCC_CAN1_CLK_ENABLE() for CAN1.
2. CAN pins configuration
   – Enable the clock for the CAN GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
   – Connect and configure the involved CAN pins using the following function HAL_GPIO_Init();

3. Initialize and configure the CAN using HAL_CAN_Init() function.
4. Transmit the desired CAN frame using HAL_CAN_Transmit() or HAL_CAN_Transmit_IT() function.
5. Receive a CAN frame using HAL_CAN_Receive() or HAL_CAN_Receive_IT() function.

Polling mode IO operation

- Start the CAN peripheral transmission and wait the end of this operation using HAL_CAN_Transmit(), at this stage user can specify the value of timeout according to his end application
- Start the CAN peripheral reception and wait the end of this operation using HAL_CAN_Receive(), at this stage user can specify the value of timeout according to his end application

Interrupt mode IO operation

- Start the CAN peripheral transmission using HAL_CAN_Transmit_IT()
- Start the CAN peripheral reception using HAL_CAN_Receive_IT()
- Use HAL_CAN_IRQHandler() called under the used CAN Interrupt subroutine
- At CAN end of transmission HAL_CAN_TxCpltCallback() function is executed and user can add his own code by customization of function pointer HAL_CAN_TxCpltCallback
- In case of CAN Error, HAL_CAN_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_CAN_ErrorCallback

CAN HAL driver macros list

Below the list of most used macros in CAN HAL driver.

- __HAL_CAN_ENABLE_IT: Enable the specified CAN interrupts
- __HAL_CAN_DISABLE_IT: Disable the specified CAN interrupts
- __HAL_CAN_GET_IT_SOURCE: Check if the specified CAN interrupt source is enabled or disabled
- __HAL_CAN_CLEAR_FLAG: Clear the CAN's pending flags
- __HAL_CAN_GET_FLAG: Get the selected CAN's flag status

You can refer to the CAN HAL driver header file for more useful macros

8.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the CAN.
- De-initialize the CAN.

This section contains the following APIs:

- *HAL_CAN_Init()*
- *HAL_CAN_ConfigFilter()*
- *HAL_CAN_DeInit()*
- *HAL_CAN_MspInit()*
- *HAL_CAN_MspDeInit()*

8.2.3 IO operation functions

This section provides functions allowing to:

- Transmit a CAN frame message.
- Receive a CAN frame message.
- Enter CAN peripheral in sleep mode.
- Wake up the CAN peripheral from sleep mode.

This section contains the following APIs:

- ***HAL_CAN_Transmit()***
- ***HAL_CAN_Transmit_IT()***
- ***HAL_CAN_Receive()***
- ***HAL_CAN_Receive_IT()***
- ***HAL_CAN_Sleep()***
- ***HAL_CAN_WakeUp()***
- ***HAL_CAN_IRQHandler()***
- ***HAL_CAN_TxCpltCallback()***
- ***HAL_CAN_RxCpltCallback()***
- ***HAL_CAN_ErrorCallback()***

8.2.4 Peripheral State and Error functions

This subsection provides functions allowing to:

- Check the CAN state.
- Check CAN Errors detected during interrupt process.

This section contains the following APIs:

- ***HAL_CAN_GetState()***
- ***HAL_CAN_GetError()***

8.2.5 Detailed description of functions

HAL_CAN_Init

Function name	**HAL_StatusTypeDef HAL_CAN_Init (CAN_HandleTypeDef * hcan)**
Function description	Initialize the CAN peripheral according to the specified parameters in the CAN_InitStruct structure and initialize the associated handle.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL:** status

HAL_CAN_ConfigFilter

Function name	**HAL_StatusTypeDef HAL_CAN_ConfigFilter (CAN_HandleTypeDef * hcan, CAN_FilterConfTypeDef * sFilterConfig)**
Function description	Configure the CAN reception filter according to the specified parameters in the CAN_FilterInitStruct.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

- **sFilterConfig:** pointer to a CAN_FilterConfTypeDef structure that contains the filter configuration information.

Return values	• **None:**

HAL_CAN_DeInit

Function name	**HAL_StatusTypeDef HAL_CAN_DeInit (CAN_HandleTypeDef * hcan)**
Function description	DeInitialize the CAN peripheral registers to their default reset values.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL:** status

HAL_CAN_MspInit

Function name	**void HAL_CAN_MspInit (CAN_HandleTypeDef * hcan)**
Function description	Initialize the CAN MSP.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **None:**

HAL_CAN_MspDeInit

Function name	**void HAL_CAN_MspDeInit (CAN_HandleTypeDef * hcan)**
Function description	DeInitialize the CAN MSP.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **None:**

HAL_CAN_Transmit

Function name	**HAL_StatusTypeDef HAL_CAN_Transmit (CAN_HandleTypeDef * hcan, uint32_t Timeout)**
Function description	Initiate and transmit a CAN frame message.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **Timeout:** Timeout duration.
Return values	• **HAL:** status

HAL_CAN_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_CAN_Transmit_IT (CAN_HandleTypeDef * hcan)**
Function description	Initiate and transmit a CAN frame message in Interrupt mode.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that

contains the configuration information for the specified CAN.

Return values	• **HAL:** status

HAL_CAN_Receive

Function name	**HAL_StatusTypeDef HAL_CAN_Receive (CAN_HandleTypeDef * hcan, uint8_t FIFONumber, uint32_t Timeout)**
Function description	Receive a correct CAN frame.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **FIFONumber:** FIFO number. • **Timeout:** Timeout duration.
Return values	• **HAL:** status

HAL_CAN_Receive_IT

Function name	**HAL_StatusTypeDef HAL_CAN_Receive_IT (CAN_HandleTypeDef * hcan, uint8_t FIFONumber)**
Function description	Receive a correct CAN frame in Interrupt mode.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **FIFONumber:** FIFO number.
Return values	• **HAL:** status

HAL_CAN_Sleep

Function name	**HAL_StatusTypeDef HAL_CAN_Sleep (CAN_HandleTypeDef * hcan)**
Function description	Enter the Sleep (low power) mode.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL:** status.

HAL_CAN_WakeUp

Function name	**HAL_StatusTypeDef HAL_CAN_WakeUp (CAN_HandleTypeDef * hcan)**
Function description	Wake up the CAN peripheral from sleep mode (after that the CAN peripheral is in the normal mode).
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL:** status.

HAL_CAN_IRQHandler

Function name	**void HAL_CAN_IRQHandler (CAN_HandleTypeDef * hcan)**

Function description	Handle CAN interrupt request.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **None:**

HAL_CAN_TxCpltCallback

Function name	**void HAL_CAN_TxCpltCallback (CAN_HandleTypeDef * hcan)**
Function description	Transmission complete callback in non-blocking mode.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **None:**

HAL_CAN_RxCpltCallback

Function name	**void HAL_CAN_RxCpltCallback (CAN_HandleTypeDef * hcan)**
Function description	Reception complete callback in non-blocking mode.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **None:**

HAL_CAN_ErrorCallback

Function name	**void HAL_CAN_ErrorCallback (CAN_HandleTypeDef * hcan)**
Function description	Error CAN callback.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **None:**

HAL_CAN_GetError

Function name	**uint32_t HAL_CAN_GetError (CAN_HandleTypeDef * hcan)**
Function description	Return the CAN error code.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **CAN:** Error Code

HAL_CAN_GetState

Function name	**HAL_CAN_StateTypeDef HAL_CAN_GetState (CAN_HandleTypeDef * hcan)**
Function description	Return the CAN handle state.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values • **HAL:** state

8.3 CAN Firmware driver defines

8.3.1 CAN

CAN Error Code

HAL_CAN_ERROR_NONE	No error
HAL_CAN_ERROR_EWG	EWG error
HAL_CAN_ERROR_EPV	EPV error
HAL_CAN_ERROR_BOF	BOF error
HAL_CAN_ERROR_STF	Stuff error
HAL_CAN_ERROR_FOR	Form error
HAL_CAN_ERROR_ACK	Acknowledgment error
HAL_CAN_ERROR_BR	Bit recessive
HAL_CAN_ERROR_BD	LEC dominant
HAL_CAN_ERROR_CRC	LEC transfer error
HAL_CAN_ERROR_FOV0	FIFO0 overrun error
HAL_CAN_ERROR_FOV1	FIFO1 overrun error

CAN Exported Constants

CAN_TXMAILBOX_0

CAN_TXMAILBOX_1

CAN_TXMAILBOX_2

CAN Exported Macros

__HAL_CAN_RESET_HANDLE_STATE

Description:

- Reset CAN handle state.

Parameters:

- __HANDLE__: CAN handle.

Return value:

- None

__HAL_CAN_ENABLE_IT

Description:

- Enable the specified CAN interrupt.

Parameters:

- __HANDLE__: CAN handle.
- __INTERRUPT__: CAN Interrupt.

Return value:

- None

__HAL_CAN_DISABLE_IT

Description:

- Disable the specified CAN interrupt.

Parameters:

- __HANDLE__: CAN handle.
- __INTERRUPT__: CAN Interrupt.

Return value:

- None

__HAL_CAN_MSG_PENDING

Description:

- Return the number of pending received messages.

Parameters:

- __HANDLE__: CAN handle.
- __FIFONUMBER__: Receive FIFO number, CAN_FIFO0 or CAN_FIFO1.

Return value:

- The: number of pending message.

__HAL_CAN_GET_FLAG

Description:

- Check whether the specified CAN flag is set or not.

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - CAN_TSR_RQCP0: Request MailBox0 Flag
  - CAN_TSR_RQCP1: Request MailBox1 Flag
  - CAN_TSR_RQCP2: Request MailBox2 Flag
  - CAN_FLAG_TXOK0: Transmission OK MailBox0 Flag
  - CAN_FLAG_TXOK1: Transmission OK MailBox1 Flag
  - CAN_FLAG_TXOK2: Transmission OK MailBox2 Flag
  - CAN_FLAG_TME0: Transmit mailbox 0 empty Flag
  - CAN_FLAG_TME1: Transmit mailbox 1 empty Flag
  - CAN_FLAG_TME2: Transmit mailbox 2 empty Flag
  - CAN_FLAG_FMP0: FIFO 0 Message Pending Flag
  - CAN_FLAG_FF0: FIFO 0 Full Flag
  - CAN_FLAG_FOV0: FIFO 0 Overrun Flag
  - CAN_FLAG_FMP1: FIFO 1 Message

Pending Flag
- CAN_FLAG_FF1: FIFO 1 Full Flag
- CAN_FLAG_FOV1: FIFO 1 Overrun Flag
- CAN_FLAG_WKU: Wake up Flag
- CAN_FLAG_SLAK: Sleep acknowledge Flag
- CAN_FLAG_SLAKI: Sleep acknowledge Flag
- CAN_FLAG_EWG: Error Warning Flag
- CAN_FLAG_EPV: Error Passive Flag
- CAN_FLAG_BOF: Bus-Off Flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_CAN_CLEAR_FLAG

Description:

- Clear the specified CAN pending flag.

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - CAN_TSR_RQCP0: Request MailBox0 Flag
  - CAN_TSR_RQCP1: Request MailBox1 Flag
  - CAN_TSR_RQCP2: Request MailBox2 Flag
  - CAN_FLAG_TXOK0: Transmission OK MailBox0 Flag
  - CAN_FLAG_TXOK1: Transmission OK MailBox1 Flag
  - CAN_FLAG_TXOK2: Transmission OK MailBox2 Flag
  - CAN_FLAG_TME0: Transmit mailbox 0 empty Flag
  - CAN_FLAG_TME1: Transmit mailbox 1 empty Flag
  - CAN_FLAG_TME2: Transmit mailbox 2 empty Flag
  - CAN_FLAG_FMP0: FIFO 0 Message Pending Flag
  - CAN_FLAG_FF0: FIFO 0 Full Flag
  - CAN_FLAG_FOV0: FIFO 0 Overrun Flag
  - CAN_FLAG_FMP1: FIFO 1 Message Pending Flag
  - CAN_FLAG_FF1: FIFO 1 Full Flag
  - CAN_FLAG_FOV1: FIFO 1 Overrun Flag

  - CAN_FLAG_WKU: Wake up Flag
  - CAN_FLAG_SLAKI: Sleep acknowledge Flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_CAN_GET_IT_SOURCE

Description:

- Check whether the specified CAN interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __INTERRUPT__: specifies the CAN interrupt source to check. This parameter can be one of the following values:
  - CAN_IT_TME: Transmit mailbox empty interrupt enable
  - CAN_IT_FMP0: FIFO0 message pending interrupt enable
  - CAN_IT_FMP1: FIFO1 message pending interrupt enable

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_CAN_TRANSMIT_STATUS

Description:

- Check the transmission status of a CAN Frame.

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __TRANSMITMAILBOX__: the number of the mailbox that is used for transmission.

Return value:

- The: new status of transmission (TRUE or FALSE).

__HAL_CAN_FIFO_RELEASE

Description:

- Release the specified receive FIFO.

Parameters:

- __HANDLE__: CAN handle.
- __FIFONUMBER__: Receive FIFO number, CAN_FIFO0 or CAN_FIFO1.

Return value:

- None

__HAL_CAN_CANCEL_TRANSMIT

Description:

- Cancel a transmit request.

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __TRANSMITMAILBOX__: the number of the mailbox that is used for transmission.

Return value:

- None

__HAL_CAN_DBG_FREEZE

Description:

- Enable or disable the DBG Freeze for CAN.

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __NEWSTATE__: new state of the CAN peripheral. This parameter can be: ENABLE (CAN reception/transmission is frozen during debug. Reception FIFO can still be accessed/controlled normally) or DISABLE (CAN is working during debug).

Return value:

- None

CAN Filter FIFO

CAN_FILTER_FIFO0	Filter FIFO 0 assignment for filter x
CAN_FILTER_FIFO1	Filter FIFO 1 assignment for filter x

CAN Filter Mode

CAN_FILTERMODE_IDMASK	Identifier mask mode
CAN_FILTERMODE_IDLIST	Identifier list mode

CAN Filter Scale

CAN_FILTERSCALE_16BIT	Two 16-bit filters
CAN_FILTERSCALE_32BIT	One 32-bit filter

CAN Flags

CAN_FLAG_RQCP0	Request MailBox0 flag
CAN_FLAG_RQCP1	Request MailBox1 flag
CAN_FLAG_RQCP2	Request MailBox2 flag
CAN_FLAG_TXOK0	Transmission OK MailBox0 flag
CAN_FLAG_TXOK1	Transmission OK MailBox1 flag
CAN_FLAG_TXOK2	Transmission OK MailBox2 flag
CAN_FLAG_TME0	Transmit mailbox 0 empty flag
CAN_FLAG_TME1	Transmit mailbox 0 empty flag
CAN_FLAG_TME2	Transmit mailbox 0 empty flag

CAN_FLAG_FF0	FIFO 0 Full flag
CAN_FLAG_FOV0	FIFO 0 Overrun flag
CAN_FLAG_FF1	FIFO 1 Full flag
CAN_FLAG_FOV1	FIFO 1 Overrun flag
CAN_FLAG_WKU	Wake up flag
CAN_FLAG_SLAK	Sleep acknowledge flag
CAN_FLAG_SLAKI	Sleep acknowledge flag
CAN_FLAG_EWG	Error warning flag
CAN_FLAG_EPV	Error passive flag
CAN_FLAG_BOF	Bus-Off flag

CAN Identifier Type

CAN_ID_STD	Standard Id
CAN_ID_EXT	Extended Id

CAN initialization Status

CAN_INITSTATUS_FAILED	CAN initialization failed
CAN_INITSTATUS_SUCCESS	CAN initialization OK

CAN Interrupts

CAN_IT_TME	Transmit mailbox empty interrupt
CAN_IT_FMP0	FIFO 0 message pending interrupt
CAN_IT_FF0	FIFO 0 full interrupt
CAN_IT_FOV0	FIFO 0 overrun interrupt
CAN_IT_FMP1	FIFO 1 message pending interrupt
CAN_IT_FF1	FIFO 1 full interrupt
CAN_IT_FOV1	FIFO 1 overrun interrupt
CAN_IT_WKU	Wake-up interrupt
CAN_IT_SLK	Sleep acknowledge interrupt
CAN_IT_EWG	Error warning interrupt
CAN_IT_EPV	Error passive interrupt
CAN_IT_BOF	Bus-off interrupt
CAN_IT_LEC	Last error code interrupt
CAN_IT_ERR	Error Interrupt

CAN Operating Mode

CAN_MODE_NORMAL	Normal mode
CAN_MODE_LOOPBACK	Loopback mode
CAN_MODE_SILENT	Silent mode
CAN_MODE_SILENT_LOOPBACK	Loopback combined with silent mode

CAN Receive FIFO Number

CAN_FIFO0	CAN FIFO 0 used to receive
CAN_FIFO1	CAN FIFO 1 used to receive

CAN Remote Transmission Request

CAN_RTR_DATA	Data frame
CAN_RTR_REMOTE	Remote frame

CAN Synchronization Jump Width

CAN_SJW_1TQ	1 time quantum
CAN_SJW_2TQ	2 time quantum
CAN_SJW_3TQ	3 time quantum
CAN_SJW_4TQ	4 time quantum

CAN Time Quantum in Bit Segment 1

CAN_BS1_1TQ	1 time quantum
CAN_BS1_2TQ	2 time quantum
CAN_BS1_3TQ	3 time quantum
CAN_BS1_4TQ	4 time quantum
CAN_BS1_5TQ	5 time quantum
CAN_BS1_6TQ	6 time quantum
CAN_BS1_7TQ	7 time quantum
CAN_BS1_8TQ	8 time quantum
CAN_BS1_9TQ	9 time quantum
CAN_BS1_10TQ	10 time quantum
CAN_BS1_11TQ	11 time quantum
CAN_BS1_12TQ	12 time quantum
CAN_BS1_13TQ	13 time quantum
CAN_BS1_14TQ	14 time quantum
CAN_BS1_15TQ	15 time quantum
CAN_BS1_16TQ	16 time quantum

CAN Time Quantum in Bit Segment 2

CAN_BS2_1TQ	1 time quantum
CAN_BS2_2TQ	2 time quantum
CAN_BS2_3TQ	3 time quantum
CAN_BS2_4TQ	4 time quantum
CAN_BS2_5TQ	5 time quantum
CAN_BS2_6TQ	6 time quantum
CAN_BS2_7TQ	7 time quantum

CAN_BS2_8TQ	8 time quantum

CAN Transmit Constants

CAN_TXSTATUS_NOMAILBOX	CAN cell did not provide CAN_TxStatus_NoMailBox

9 HAL CORTEX Generic Driver

9.1 CORTEX Firmware driver registers structures

9.1.1 MPU_Region_InitTypeDef

Data Fields

- ***uint8_t Enable***
- ***uint8_t Number***
- ***uint32_t BaseAddress***
- ***uint8_t Size***
- ***uint8_t SubRegionDisable***
- ***uint8_t TypeExtField***
- ***uint8_t AccessPermission***
- ***uint8_t DisableExec***
- ***uint8_t IsShareable***
- ***uint8_t IsCacheable***
- ***uint8_t IsBufferable***

Field Documentation

- ***uint8_t MPU_Region_InitTypeDef::Enable***
  Specifies the status of the region. This parameter can be a value of ***CORTEX_MPU_Region_Enable***
- ***uint8_t MPU_Region_InitTypeDef::Number***
  Specifies the number of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Number***
- ***uint32_t MPU_Region_InitTypeDef::BaseAddress***
  Specifies the base address of the region to protect.
- ***uint8_t MPU_Region_InitTypeDef::Size***
  Specifies the size of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Size***
- ***uint8_t MPU_Region_InitTypeDef::SubRegionDisable***
  Specifies the number of the subregion protection to disable. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint8_t MPU_Region_InitTypeDef::TypeExtField***
  Specifies the TEX field level. This parameter can be a value of ***CORTEX_MPU_TEX_Levels***
- ***uint8_t MPU_Region_InitTypeDef::AccessPermission***
  Specifies the region access permission type. This parameter can be a value of ***CORTEX_MPU_Region_Permission_Attributes***
- ***uint8_t MPU_Region_InitTypeDef::DisableExec***
  Specifies the instruction access status. This parameter can be a value of ***CORTEX_MPU_Instruction_Access***
- ***uint8_t MPU_Region_InitTypeDef::IsShareable***
  Specifies the shareability status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Shareable***
- ***uint8_t MPU_Region_InitTypeDef::IsCacheable***
  Specifies the cacheable status of the region protected. This parameter can be a value of ***CORTEX_MPU_Access_Cacheable***

- ***uint8_t MPU_Region_InitTypeDef::IsBufferable***
  Specifies the bufferable status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Bufferable***

9.2 CORTEX Firmware driver API description

9.2.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

This section provides functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M4 exceptions are managed by CMSIS functions.

1. Configure the NVIC Priority Grouping using HAL_NVIC_SetPriorityGrouping() function.
2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority().
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ(). When the NVIC_PRIORITYGROUP_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the sub priority. IRQ priority order (sorted by highest to lowest priority): Lowest pre-emption priorityLowest sub priorityLowest hardware priority (IRQ number)

How to configure SysTick using CORTEX HAL driver

Setup SysTick Timer for time base.

- The HAL_SYSTICK_Config() function calls the SysTick_Config() function which is a CMSIS function that:
  - Configures the SysTick Reload register with value passed as function parameter.
  - Configures the SysTick IRQ priority to the lowest value (0x0F).
  - Resets the SysTick Counter register.
  - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
  - Enables the SysTick Interrupt.
  - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK_Div8 by calling the macro __HAL_CORTEX_SYSTICKCLK_CONFIG(SYSTICK_CLKSOURCE_HCLK_DIV8) just after the HAL_SYSTICK_Config() function call. The __HAL_CORTEX_SYSTICKCLK_CONFIG() macro is defined inside the stm32l4xx_hal_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL_NVIC_SetPriority(SysTick_IRQn,...) function just after the HAL_SYSTICK_Config() function call. The HAL_NVIC_SetPriority() call the NVIC_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL_SYSTICK_Config() function
  - Reload Value should not exceed 0xFFFFFF

9.2.2 Initialization and Configuration functions

This section provides the CORTEX HAL driver functions allowing to configure Interrupts SysTick functionalities

This section contains the following APIs:

- *HAL_NVIC_SetPriorityGrouping()*
- *HAL_NVIC_SetPriority()*
- *HAL_NVIC_EnableIRQ()*
- *HAL_NVIC_DisableIRQ()*
- *HAL_NVIC_SystemReset()*
- *HAL_SYSTICK_Config()*
- *HAL_MPU_Disable()*
- *HAL_MPU_Enable()*

9.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to control the CORTEX (NVIC, SYSTICK, MPU) functionalities.

This section contains the following APIs:

- *HAL_NVIC_GetPriorityGrouping()*
- *HAL_NVIC_GetPriority()*
- *HAL_NVIC_SetPendingIRQ()*
- *HAL_NVIC_GetPendingIRQ()*
- *HAL_NVIC_ClearPendingIRQ()*
- *HAL_NVIC_GetActive()*
- *HAL_SYSTICK_CLKSourceConfig()*
- *HAL_SYSTICK_IRQHandler()*
- *HAL_SYSTICK_Callback()*
- *HAL_MPU_ConfigRegion()*

9.2.4 Detailed description of functions

HAL_NVIC_SetPriorityGrouping

Function name	**void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)**
Function description	Set the priority grouping field (pre-emption priority and subpriority) using the required unlock sequence.
Parameters	• **PriorityGroup:** The priority grouping bits length. This parameter can be one of the following values: – NVIC_PRIORITYGROUP_0: 0 bit for pre-emption priority, 4 bits for subpriority – NVIC_PRIORITYGROUP_1: 1 bit for pre-emption priority, 3 bits for subpriority – NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority, 2 bits for subpriority – NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority, 1 bit for subpriority – NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority, 0 bit for subpriority
Return values	• **None:**
Notes	• When the NVIC_PriorityGroup_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the subpriority.

HAL_NVIC_SetPriority

Function name	**void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)**
Function description	Set the priority of an interrupt.
Parameters	• **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l4xxxx.h)) • **PreemptPriority:** The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority • **SubPriority:** the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.
Return values	• **None:**

HAL_NVIC_EnableIRQ

Function name	**void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)**
Function description	Enable a device specific interrupt in the NVIC interrupt controller.
Parameters	• **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l4xxxx.h))
Return values	• **None:**
Notes	• To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

HAL_NVIC_DisableIRQ

Function name	**void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)**
Function description	Disable a device specific interrupt in the NVIC interrupt controller.
Parameters	• **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l4xxxx.h))
Return values	• **None:**

HAL_NVIC_SystemReset

Function name	**void HAL_NVIC_SystemReset (void)**
Function description	Initiate a system reset request to reset the MCU.
Return values	• **None:**

HAL_SYSTICK_Config

Function name	**uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)**

Function description	Initialize the System Timer with interrupt enabled and start the System Tick Timer (SysTick): Counter is in free running mode to generate periodic interrupts.
Parameters	• **TicksNumb:** Specifies the ticks Number of ticks between two interrupts.
Return values	• **status:** - 0 Function succeeded. – 1 Function failed.

HAL_MPU_Disable

Function name	**__STATIC_INLINE void HAL_MPU_Disable (void)**
Function description	Disable the MPU.
Return values	• **None:**

HAL_MPU_Enable

Function name	**__STATIC_INLINE void HAL_MPU_Enable (uint32_t MPU_Control)**
Function description	Enable the MPU.
Parameters	• **MPU_Control:** Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged accessto the default memory This parameter can be one of the following values: – MPU_HFNMI_PRIVDEF_NONE – MPU_HARDFAULT_NMI – MPU_PRIVILEGED_DEFAULT – MPU_HFNMI_PRIVDEF
Return values	• **None:**

HAL_NVIC_GetPriorityGrouping

Function name	**uint32_t HAL_NVIC_GetPriorityGrouping (void)**
Function description	Get the priority grouping field from the NVIC Interrupt Controller.
Return values	• **Priority:** grouping field (SCB->AIRCR [10:8] PRIGROUP field)

HAL_NVIC_GetPriority

Function name	**void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority, uint32_t * pSubPriority)**
Function description	Get the priority of an interrupt.
Parameters	• **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l4xxxx.h)) • **PriorityGroup:** the priority grouping bits length. This parameter can be one of the following values:

  - NVIC_PRIORITYGROUP_0: 0 bit for pre-emption priority, 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1: 1 bit for pre-emption priority, 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority, 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority, 1 bit for subpriority
  - NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority, 0 bit for subpriority
- **pPreemptPriority:** Pointer on the Preemptive priority value (starting from 0).
- **pSubPriority:** Pointer on the Subpriority value (starting from 0).

Return values	• **None:**

HAL_NVIC_GetPendingIRQ

Function name	**uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)**
Function description	Get Pending Interrupt (read the pending register in the NVIC and return the pending bit for the specified interrupt).
Parameters	• **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l4xxxx.h))
Return values	• **status:** - 0 Interrupt status is not pending. – 1 Interrupt status is pending.

HAL_NVIC_SetPendingIRQ

Function name	**void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)**
Function description	Set Pending bit of an external interrupt.
Parameters	• **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l4xxxx.h))
Return values	• **None:**

HAL_NVIC_ClearPendingIRQ

Function name	**void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)**
Function description	Clear the pending bit of an external interrupt.
Parameters	• **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l4xxxx.h))
Return values	• **None:**

HAL_NVIC_GetActive

Function name	**uint32_t HAL_NVIC_GetActive (IRQn_Type IRQn)**
Function description	Get active interrupt (read the active register in NVIC and return the active bit).
Parameters	• **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l4xxxx.h))
Return values	• **status:** - 0 Interrupt status is not pending. – 1 Interrupt status is pending.

HAL_SYSTICK_CLKSourceConfig

Function name	**void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)**
Function description	Configure the SysTick clock source.
Parameters	• **CLKSource:** specifies the SysTick clock source. This parameter can be one of the following values: – SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source. – SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.
Return values	• **None:**

HAL_SYSTICK_IRQHandler

Function name	**void HAL_SYSTICK_IRQHandler (void)**
Function description	Handle SYSTICK interrupt request.
Return values	• **None:**

HAL_SYSTICK_Callback

Function name	**void HAL_SYSTICK_Callback (void)**
Function description	SYSTICK callback.
Return values	• **None:**

HAL_MPU_ConfigRegion

Function name	**void HAL_MPU_ConfigRegion (MPU_Region_InitTypeDef * MPU_Init)**
Function description	Initialize and configure the Region and the memory to be protected.
Parameters	• **MPU_Init:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.
Return values	• **None:**

9.3 CORTEX Firmware driver defines

9.3.1 CORTEX

CORTEX MPU Instruction Access Bufferable

MPU_ACCESS_BUFFERABLE

MPU_ACCESS_NOT_BUFFERABLE

CORTEX MPU Instruction Access Cacheable

MPU_ACCESS_CACHEABLE

MPU_ACCESS_NOT_CACHEABLE

CORTEX MPU Instruction Access Shareable

MPU_ACCESS_SHAREABLE

MPU_ACCESS_NOT_SHAREABLE

CORTEX MPU HFNMI and PRIVILEGED Access control

MPU_HFNMI_PRIVDEF_NONE

MPU_HARDFAULT_NMI

MPU_PRIVILEGED_DEFAULT

MPU_HFNMI_PRIVDEF

CORTEX MPU Instruction Access

MPU_INSTRUCTION_ACCESS_ENABLE

MPU_INSTRUCTION_ACCESS_DISABLE

CORTEX MPU Region Enable

MPU_REGION_ENABLE

MPU_REGION_DISABLE

CORTEX MPU Region Number

MPU_REGION_NUMBER0

MPU_REGION_NUMBER1

MPU_REGION_NUMBER2

MPU_REGION_NUMBER3

MPU_REGION_NUMBER4

MPU_REGION_NUMBER5

MPU_REGION_NUMBER6

MPU_REGION_NUMBER7

CORTEX MPU Region Permission Attributes

MPU_REGION_NO_ACCESS

MPU_REGION_PRIV_RW

MPU_REGION_PRIV_RW_URO

MPU_REGION_FULL_ACCESS

MPU_REGION_PRIV_RO

MPU_REGION_PRIV_RO_URO

CORTEX MPU Region Size

MPU_REGION_SIZE_32B

MPU_REGION_SIZE_64B

MPU_REGION_SIZE_128B

MPU_REGION_SIZE_256B

MPU_REGION_SIZE_512B

MPU_REGION_SIZE_1KB

MPU_REGION_SIZE_2KB

MPU_REGION_SIZE_4KB

MPU_REGION_SIZE_8KB

MPU_REGION_SIZE_16KB

MPU_REGION_SIZE_32KB

MPU_REGION_SIZE_64KB

MPU_REGION_SIZE_128KB

MPU_REGION_SIZE_256KB

MPU_REGION_SIZE_512KB

MPU_REGION_SIZE_1MB

MPU_REGION_SIZE_2MB

MPU_REGION_SIZE_4MB

MPU_REGION_SIZE_8MB

MPU_REGION_SIZE_16MB

MPU_REGION_SIZE_32MB

MPU_REGION_SIZE_64MB

MPU_REGION_SIZE_128MB

MPU_REGION_SIZE_256MB

MPU_REGION_SIZE_512MB

MPU_REGION_SIZE_1GB

MPU_REGION_SIZE_2GB

MPU_REGION_SIZE_4GB

CORTEX MPU TEX Levels

MPU_TEX_LEVEL0

MPU_TEX_LEVEL1

MPU_TEX_LEVEL2

CORTEX Preemption Priority Group

NVIC_PRIORITYGROUP_0	0 bit for pre-emption priority, 4 bits for subpriority
NVIC_PRIORITYGROUP_1	1 bit for pre-emption priority, 3 bits for subpriority
NVIC_PRIORITYGROUP_2	2 bits for pre-emption priority, 2 bits for subpriority
NVIC_PRIORITYGROUP_3	3 bits for pre-emption priority, 1 bit for subpriority
NVIC_PRIORITYGROUP_4	4 bits for pre-emption priority, 0 bit for subpriority

CORTEX SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

10 HAL CRC Generic Driver

10.1 CRC Firmware driver registers structures

10.1.1 CRC_InitTypeDef

Data Fields

- ***uint8_t DefaultPolynomialUse***
- ***uint8_t DefaultInitValueUse***
- ***uint32_t GeneratingPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t InitValue***
- ***uint32_t InputDataInversionMode***
- ***uint32_t OutputDataInversionMode***

Field Documentation

- ***uint8_t CRC_InitTypeDef::DefaultPolynomialUse***
  This parameter is a value of ***CRC_Default_Polynomial*** and indicates if default polynomial is used. If set to DEFAULT_POLYNOMIAL_ENABLE, resort to default X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1. In that case, there is no need to set GeneratingPolynomial field. If otherwise set to DEFAULT_POLYNOMIAL_DISABLE, GeneratingPolynomial and CRCLength fields must be set.
- ***uint8_t CRC_InitTypeDef::DefaultInitValueUse***
  This parameter is a value of ***CRC_Default_InitValue_Use*** and indicates if default init value is used. If set to DEFAULT_INIT_VALUE_ENABLE, resort to default 0xFFFFFFFF value. In that case, there is no need to set InitValue field. If otherwise set to DEFAULT_INIT_VALUE_DISABLE, InitValue field must be set.
- ***uint32_t CRC_InitTypeDef::GeneratingPolynomial***
  Set CRC generating polynomial as a 7, 8, 16 or 32-bit long value for a polynomial degree respectively equal to 7, 8, 16 or 32. This field is written in normal representation, e.g., for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65. No need to specify it if DefaultPolynomialUse is set to DEFAULT_POLYNOMIAL_ENABLE.
- ***uint32_t CRC_InitTypeDef::CRCLength***
  This parameter is a value of ***CRC_Polynomial_Sizes*** and indicates CRC length. Value can be either one of **CRC_POLYLENGTH_32B** (32-bit CRC), **CRC_POLYLENGTH_16B** (16-bit CRC), **CRC_POLYLENGTH_8B** (8-bit CRC), **CRC_POLYLENGTH_7B** (7-bit CRC).
- ***uint32_t CRC_InitTypeDef::InitValue***
  Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT_INIT_VALUE_ENABLE.
- ***uint32_t CRC_InitTypeDef::InputDataInversionMode***
  This parameter is a value of ***CRCEx_Input_Data_Inversion*** and specifies input data inversion mode. Can be either one of the following values **CRC_INPUTDATA_INVERSION_NONE** no input data inversion **CRC_INPUTDATA_INVERSION_BYTE** byte-wise inversion, 0x1A2B3C4D becomes 0x58D43CB2 **CRC_INPUTDATA_INVERSION_HALFWORD** halfword-wise inversion, 0x1A2B3C4D becomes 0xD458B23C **CRC_INPUTDATA_INVERSION_WORD** word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458

- ***uint32_t CRC_InitTypeDef::OutputDataInversionMode***
  This parameter is a value of ***CRCEx_Output_Data_Inversion*** and specifies output data (i.e. CRC) inversion mode. Can be either **CRC_OUTPUTDATA_INVERSION_DISABLE** no CRC inversion, **CRC_OUTPUTDATA_INVERSION_ENABLE** CRC 0x11223344 is converted into 0x22CC4488

10.1.2 CRC_HandleTypeDef

Data Fields

- ***CRC_TypeDef * Instance***
- ***CRC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRC_StateTypeDef State***
- ***uint32_t InputDataFormat***

Field Documentation

- ***CRC_TypeDef* CRC_HandleTypeDef::Instance***
  Register base address
- ***CRC_InitTypeDef CRC_HandleTypeDef::Init***
  CRC configuration parameters
- ***HAL_LockTypeDef CRC_HandleTypeDef::Lock***
  CRC Locking object
- ***__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State***
  CRC communication state
- ***uint32_t CRC_HandleTypeDef::InputDataFormat***
  This parameter is a value of ***CRC_Input_Buffer_Format*** and specifies input data format. Can be either **CRC_INPUTDATA_FORMAT_BYTES** input data is a stream of bytes (8-bit data) **CRC_INPUTDATA_FORMAT_HALFWORDS** input data is a stream of half-words (16-bit data) **CRC_INPUTDATA_FORMAT_WORDS** input data is a stream of words (32-bit data) Note that constant CRC_INPUT_FORMAT_UNDEFINED is defined but an initialization error must occur if InputBufferFormat is not one of the three values listed above

10.2 CRC Firmware driver API description

10.2.1 How to use this driver

- Enable CRC AHB clock using __HAL_RCC_CRC_CLK_ENABLE();
- Initialize CRC calculator
  - specify generating polynomial (IP default or non-default one)
  - specify initialization value (IP default or non-default one)
  - specify input data format
  - specify input or output data inversion mode if any
- Use HAL_CRC_Accumulate() function to compute the CRC value of the input data buffer starting with the previously computed CRC as initialization value
- Use HAL_CRC_Calculate() function to compute the CRC value of the input data buffer starting with the defined initialization value (default or non-default) to initiate CRC calculation

10.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle
- DeInitialize the CRC peripheral
- Initialize the CRC MSP (MCU Specific Package)
- DeInitialize the CRC MSP

This section contains the following APIs:

- ***HAL_CRC_Init()***
- ***HAL_CRC_DeInit()***
- ***HAL_CRC_MspInit()***
- ***HAL_CRC_MspDeInit()***

10.2.3 Peripheral Control functions

This section provides functions allowing to:

- compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer using the combination of the previous CRC value and the new one

or

- compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer independently of the previous CRC value.

This section contains the following APIs:

- ***HAL_CRC_Accumulate()***
- ***HAL_CRC_Calculate()***

10.2.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- ***HAL_CRC_GetState()***

10.2.5 Detailed description of functions

HAL_CRC_Init

Function name	**HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)**
Function description	Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle.
Parameters	• **hcrc:** CRC handle
Return values	• **HAL:** status

HAL_CRC_DeInit

Function name	**HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)**
Function description	DeInitialize the CRC peripheral.
Parameters	• **hcrc:** CRC handle
Return values	• **HAL:** status

HAL_CRC_MspInit

Function name	**void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)**
Function description	Initializes the CRC MSP.
Parameters	• **hcrc:** CRC handle
Return values	• **None:**

HAL_CRC_MspDeInit

Function name	**void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)**
Function description	DeInitialize the CRC MSP.
Parameters	• **hcrc:** CRC handle
Return values	• **None:**

HAL_CRC_Accumulate

Function name	**uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)**
Function description	Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with the previously computed CRC as initialization value.
Parameters	• **hcrc:** CRC handle • **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat. • **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).
Return values	• **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)
Notes	• By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_Calculate

Function name	**uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)**
Function description	Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.
Parameters	• **hcrc:** CRC handle • **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat. • **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values	•	**uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)
Notes	•	By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_GetState

Function name	**HAL_CRC_StateTypeDef HAL_CRC_GetState (CRC_HandleTypeDef * hcrc)**
Function description	Return the CRC handle state.
Parameters	• **hcrc:** CRC handle
Return values	• **HAL:** state

10.3 CRC Firmware driver defines

10.3.1 CRC

CRC API aliases

HAL_CRC_Input_Data_Reverse	Aliased to HAL_CRCEx_Input_Data_Reverse for inter STM32 series compatibility
HAL_CRC_Output_Data_Reverse	Aliased to HAL_CRCEx_Output_Data_Reverse for inter STM32 series compatibility

Default CRC computation initialization value

DEFAULT_CRC_INITVALUE	Initial CRC default value

Indicates whether or not default init value is used

DEFAULT_INIT_VALUE_ENABLE	Enable initial CRC default value
DEFAULT_INIT_VALUE_DISABLE	Disable initial CRC default value

Indicates whether or not default polynomial is used

DEFAULT_POLYNOMIAL_ENABLE	Enable default generating polynomial 0x04C11DB7
DEFAULT_POLYNOMIAL_DISABLE	Disable default generating polynomial 0x04C11DB7

Default CRC generating polynomial

DEFAULT_CRC32_POLY	X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1

CRC Exported Macros

__HAL_CRC_RESET_HANDLE_STATE

Description:

- Reset CRC handle state.

Parameters:

- __HANDLE__: CRC handle.

Return value:

- None

__HAL_CRC_DR_RESET

Description:

- Reset CRC Data Register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_INITIALCRCVALUE_CONFIG

Description:

- Set CRC INIT non-default value.

Parameters:

- __HANDLE__: CRC handle
- __INIT__: 32-bit initial value

Return value:

- None

__HAL_CRC_SET_IDR

Description:

- Store a 8-bit data in the Independent Data(ID) register.

Parameters:

- __HANDLE__: CRC handle
- __VALUE__: 8-bit value to be stored in the ID register

Return value:

- None

__HAL_CRC_GET_IDR

Description:

- Return the 8-bit data stored in the Independent Data(ID) register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- 8-bit: value of the ID register

Input Buffer Format

CRC_INPUTDATA_FORMAT_UNDEFINED	Undefined input data format
CRC_INPUTDATA_FORMAT_BYTES	Input data in byte format
CRC_INPUTDATA_FORMAT_HALFWORDS	Input data in half-word format
CRC_INPUTDATA_FORMAT_WORDS	Input data in word format

Polynomial sizes to configure the IP

CRC_POLYLENGTH_32B	Resort to a 32-bit long generating polynomial

CRC_POLYLENGTH_16B	Resort to a 16-bit long generating polynomial
CRC_POLYLENGTH_8B	Resort to a 8-bit long generating polynomial
CRC_POLYLENGTH_7B	Resort to a 7-bit long generating polynomial

CRC polynomial possible sizes actual definitions

HAL_CRC_LENGTH_32B	32-bit long CRC
HAL_CRC_LENGTH_16B	16-bit long CRC
HAL_CRC_LENGTH_8B	8-bit long CRC
HAL_CRC_LENGTH_7B	7-bit long CRC

11 HAL CRC Extension Driver

11.1 CRCEx Firmware driver API description

11.1.1 How to use this driver

- Set user-defined generating polynomial thru HAL_CRCEx_Polynomial_Set()
- Configure Input or Output data inversion

11.1.2 Extended configuration functions

This section provides functions allowing to:

- Configure the generating polynomial
- Configure the input data inversion
- Configure the output data inversion

This section contains the following APIs:

- ***HAL_CRCEx_Polynomial_Set()***
- ***HAL_CRCEx_Input_Data_Reverse()***
- ***HAL_CRCEx_Output_Data_Reverse()***

11.1.3 Detailed description of functions

HAL_CRCEx_Polynomial_Set

Function name	**HAL_StatusTypeDef HAL_CRCEx_Polynomial_Set (CRC_HandleTypeDef * hcrc, uint32_t Pol, uint32_t PolyLength)**
Function description	Initialize the CRC polynomial if different from default one.
Parameters	• **hcrc:** CRC handle • **Pol:** CRC generating polynomial (7, 8, 16 or 32-bit long). This parameter is written in normal representation, e.g. – for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65 – for a polynomial of degree 16, X^16 + X^12 + X^5 + 1 is written 0x1021 • **PolyLength:** CRC polynomial length. This parameter can be one of the following values: – CRC_POLYLENGTH_7B 7-bit long CRC (generating polynomial of degree 7) – CRC_POLYLENGTH_8B 8-bit long CRC (generating polynomial of degree 8) – CRC_POLYLENGTH_16B 16-bit long CRC (generating polynomial of degree 16) – CRC_POLYLENGTH_32B 32-bit long CRC (generating polynomial of degree 32)
Return values	• **HAL:** status

HAL_CRCEx_Input_Data_Reverse

Function name	**HAL_StatusTypeDef HAL_CRCEx_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)**
Function description	Set the Reverse Input data mode.
Parameters	• **hcrc:** CRC handle • **InputReverseMode:** Input Data inversion mode. This parameter can be one of the following values: – CRC_INPUTDATA_INVERSION_NONE no change in bit order (default value) – CRC_INPUTDATA_INVERSION_BYTE Byte-wise bit reversal – CRC_INPUTDATA_INVERSION_HALFWORD HalfWord-wise bit reversal – CRC_INPUTDATA_INVERSION_WORD Word-wise bit reversal
Return values	• **HAL:** status

HAL_CRCEx_Output_Data_Reverse

Function name	**HAL_StatusTypeDef HAL_CRCEx_Output_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t OutputReverseMode)**
Function description	Set the Reverse Output data mode.
Parameters	• **hcrc:** CRC handle • **OutputReverseMode:** Output Data inversion mode. This parameter can be one of the following values: – CRC_OUTPUTDATA_INVERSION_DISABLE no CRC inversion (default value) – CRC_OUTPUTDATA_INVERSION_ENABLE bit-level inversion (e.g. for a 8-bit CRC: 0xB5 becomes 0xAD)
Return values	• **HAL:** status

11.2 CRCEx Firmware driver defines

11.2.1 CRCEx

CRCEx Exported Macros

__HAL_CRC_OUTPUTREVERSAL_ENABLE	**Description:** • Set CRC output reversal. **Parameters:** • __HANDLE__: CRC handle **Return value:** • None
__HAL_CRC_OUTPUTREVERSAL_DISABLE	**Description:** • Unset CRC output reversal. **Parameters:**

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_POLYNOMIAL_CONFIG

Description:

- Set CRC non-default polynomial.

Parameters:

- __HANDLE__: CRC handle
- __POLYNOMIAL__: 7, 8, 16 or 32-bit polynomial

Return value:

- None

Input Data Inversion Modes

CRC_INPUTDATA_INVERSION_NONE	No input data inversion
CRC_INPUTDATA_INVERSION_BYTE	Byte-wise input data inversion
CRC_INPUTDATA_INVERSION_HALFWORD	HalfWord-wise input data inversion
CRC_INPUTDATA_INVERSION_WORD	Word-wise input data inversion

Output Data Inversion Modes

CRC_OUTPUTDATA_INVERSION_DISABLE	No output data inversion
CRC_OUTPUTDATA_INVERSION_ENABLE	Bit-wise output data inversion

12 HAL CRYP Generic Driver

12.1 CRYP Firmware driver registers structures

12.1.1 CRYP_InitTypeDef

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint32_t OperatingMode***
- ***uint32_t ChainingMode***
- ***uint32_t KeyWriteFlag***
- ***uint32_t GCMCMACPhase***
- ***uint8_t * pKey***
- ***uint8_t * pInitVect***
- ***uint8_t * Header***
- ***uint64_t HeaderSize***

Field Documentation

- ***uint32_t CRYP_InitTypeDef::DataType***
  32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of ***CRYP_Data_Type***
- ***uint32_t CRYP_InitTypeDef::KeySize***
  128 or 256-bit key length. This parameter can be a value of ***CRYP_Key_Size***
- ***uint32_t CRYP_InitTypeDef::OperatingMode***
  AES operating mode. This parameter can be a value of ***CRYP_AES_OperatingMode***
- ***uint32_t CRYP_InitTypeDef::ChainingMode***
  AES chaining mode. This parameter can be a value of ***CRYP_AES_ChainingMode***
- ***uint32_t CRYP_InitTypeDef::KeyWriteFlag***
  Allows to bypass or not key write-up before decryption. This parameter can be a value of ***CRYP_Key_Write***
- ***uint32_t CRYP_InitTypeDef::GCMCMACPhase***
  Indicates the processing phase of the Galois Counter Mode (GCM), Galois Message Authentication Code (GMAC), Cipher Message Authentication Code (CMAC) (when applicable) or Counter with Cipher Mode (CCM) (when applicable). This parameter can be a value of ***CRYP_GCM_CMAC_Phase***
- ***uint8_t* CRYP_InitTypeDef::pKey***
  Encryption/Decryption Key
- ***uint8_t* CRYP_InitTypeDef::pInitVect***
  Initialization Vector used for CTR, CBC, GCM/GMAC, CMAC (when applicable) and CCM (when applicable) modes
- ***uint8_t* CRYP_InitTypeDef::Header***
  Header used in GCM/GMAC, CMAC (when applicable) and CCM (when applicable) modes
- ***uint64_t CRYP_InitTypeDef::HeaderSize***
  Header size in bytes

12.1.2 CRYP_HandleTypeDef

Data Fields

- ***AES_TypeDef * Instance***

- ***CRYP_InitTypeDef Init***
- ***uint8_t * pCrypInBuffPtr***
- ***uint8_t * pCrypOutBuffPtr***
- ***uint32_t CrypInCount***
- ***uint32_t CrypOutCount***
- ***HAL_PhaseTypeDef Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRYP_STATETypeDef State***
- ***__IO uint32_t ErrorCode***
- ***HAL_SuspendTypeDef SuspendRequest***

Field Documentation

- ***AES_TypeDef* CRYP_HandleTypeDef::Instance***
  Register base address
- ***CRYP_InitTypeDef CRYP_HandleTypeDef::Init***
  CRYP initialization parameters
- ***uint8_t* CRYP_HandleTypeDef::pCrypInBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) input buffer
- ***uint8_t* CRYP_HandleTypeDef::pCrypOutBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) output buffer
- ***uint32_t CRYP_HandleTypeDef::CrypInCount***
  Input data size in bytes or, after suspension, the remaining number of bytes to process
- ***uint32_t CRYP_HandleTypeDef::CrypOutCount***
  Output data size in bytes
- ***HAL_PhaseTypeDef CRYP_HandleTypeDef::Phase***
  CRYP peripheral processing phase for GCM, GMAC, CMAC (when applicable) or CCM (when applicable) modes. Indicates the last phase carried out to ease phase transitions
- ***DMA_HandleTypeDef* CRYP_HandleTypeDef::hdmain***
  CRYP peripheral Input DMA handle parameters
- ***DMA_HandleTypeDef* CRYP_HandleTypeDef::hdmaout***
  CRYP peripheral Output DMA handle parameters
- ***HAL_LockTypeDef CRYP_HandleTypeDef::Lock***
  CRYP locking object
- ***__IO HAL_CRYP_STATETypeDef CRYP_HandleTypeDef::State***
  CRYP peripheral state
- ***__IO uint32_t CRYP_HandleTypeDef::ErrorCode***
  CRYP peripheral error code
- ***HAL_SuspendTypeDef CRYP_HandleTypeDef::SuspendRequest***
  CRYP peripheral suspension request flag

12.2 CRYP Firmware driver API description

12.2.1 How to use this driver

The CRYP HAL driver can be used as follows:

1. Initialize the CRYP low level resources by implementing the HAL_CRYP_MspInit():
   - Enable the CRYP interface clock using __HAL_RCC_AES_CLK_ENABLE()
   - In case of using interrupts (e.g. HAL_CRYP_AES_IT())
     - Configure the CRYP interrupt priority using HAL_NVIC_SetPriority()
     - Enable the AES IRQ handler using HAL_NVIC_EnableIRQ()

        - In AES IRQ handler, call HAL_CRYP_IRQHandler()
    - In case of using DMA to control data transfer (e.g. HAL_CRYPEx_AES_DMA())
        - Enable the DMA2 interface clock using __HAL_RCC_DMA2_CLK_ENABLE()
        - Configure and enable two DMA channels one for managing data transfer from memory to peripheral (input channel) and another channel for managing data transfer from peripheral to memory (output channel)
        - Associate the initialized DMA handle to the CRYP DMA handle using __HAL_LINKDMA()
        - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA channels. The output channel should have higher priority than the input channel. Resort to HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the CRYP HAL using HAL_CRYP_Init(). This function configures:
    - The data type: 1-bit, 8-bit, 16-bit and 32-bit
    - The AES operating mode (encryption, key derivation and/or decryption)
    - The AES chaining mode (ECB, CBC, CTR, GCM, GMAC, CMAC when applicable, CCM when applicable)
    - The encryption/decryption key if so required
    - The initialization vector or nonce if applicable (not used in ECB mode).
3. Three processing (encryption/decryption) functions are available:
    - Polling mode: encryption and decryption APIs are blocking functions i.e. they process the data and wait till the processing is finished
    - Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt
    - DMA mode: encryption and decryption APIs are not blocking functions i.e. the data transfer is ensured by DMA
4. Call HAL_CRYP_DeInit() to deinitialize the CRYP peripheral.

12.2.2 Initialization and deinitialization functions

This section provides functions allowing to:

- Initialize the CRYP according to the specified parameters in the CRYP_InitTypeDef and creates the associated handle
- DeInitialize the CRYP peripheral
- Initialize the CRYP MSP (MCU Specific Package)
- De-Initialize the CRYP MSP

Specific care must be taken to format the key and the Initialization Vector IV!

If the key is defined as a 128-bit long array key[127..0] = {b127 ... b0} where b127 is the MSB and b0 the LSB, the key must be stored in MCU memory

- as a sequence of words where the MSB word comes first (occupies the lowest memory address)
- where each word is byte-swapped:
    - address n+0: 0b b103 .. b96 b111 .. b104 b119 .. b112 b127 .. b120
    - address n+4: 0b b71 .. b64 b79 .. b72 b87 .. b80 b95 .. b88
    - address n+8: 0b b39 .. b32 b47 .. b40 b55 .. b48 b63 .. b56
    - address n+C: 0b b7 .. b0 b15 .. b8 b23 .. b16 b31 .. b24

Hereafter, another illustration when considering a 128-bit long key made of 16 bytes {B15..B0}. The 4 32-bit words that make the key must be stored as follows in MCU memory:

- address n+0: 0x B12 B13 B14 B15
- address n+4: 0x B8 B9 B10 B11
- address n+8: 0x B4 B5 B6 B7
- address n+C: 0x B0 B1 B2 B3

which leads to the expected setting

- AES_KEYR3 = 0x B15 B14 B13 B12
- AES_KEYR2 = 0x B11 B10 B9 B8
- AES_KEYR1 = 0x B7 B6 B5 B4
- AES_KEYR0 = 0x B3 B2 B1 B0

Same format must be applied for a 256-bit long key made of 32 bytes {B31..B0}. The 8 32-bit words that make the key must be stored as follows in MCU memory:

- address n+00: 0x B28 B29 B30 B31
- address n+04: 0x B24 B25 B26 B27
- address n+08: 0x B20 B21 B22 B23
- address n+0C: 0x B16 B17 B18 B19
- address n+10: 0x B12 B13 B14 B15
- address n+14: 0x B8 B9 B10 B11
- address n+18: 0x B4 B5 B6 B7
- address n+1C: 0x B0 B1 B2 B3

which leads to the expected setting

- AES_KEYR7 = 0x B31 B30 B29 B28
- AES_KEYR6 = 0x B27 B26 B25 B24
- AES_KEYR5 = 0x B23 B22 B21 B20
- AES_KEYR4 = 0x B19 B18 B17 B16
- AES_KEYR3 = 0x B15 B14 B13 B12
- AES_KEYR2 = 0x B11 B10 B9 B8
- AES_KEYR1 = 0x B7 B6 B5 B4
- AES_KEYR0 = 0x B3 B2 B1 B0

Initialization Vector IV (4 32-bit words) format must follow the same as that of a 128-bit long key.

This section contains the following APIs:

- ***HAL_CRYP_Init()***
- ***HAL_CRYP_DeInit()***
- ***HAL_CRYP_MspInit()***
- ***HAL_CRYP_MspDeInit()***

12.2.3 AES processing functions

This section provides functions allowing to:

- Encrypt plaintext using AES algorithm in different chaining modes
- Decrypt cyphertext using AES algorithm in different chaining modes

Three processing functions are available:

- Polling mode

- Interrupt mode
- DMA mode

This section contains the following APIs:

- *HAL_CRYP_AESECB_Encrypt()*
- *HAL_CRYP_AESCBC_Encrypt()*
- *HAL_CRYP_AESCTR_Encrypt()*
- *HAL_CRYP_AESECB_Decrypt()*
- *HAL_CRYP_AESCBC_Decrypt()*
- *HAL_CRYP_AESCTR_Decrypt()*
- *HAL_CRYP_AESECB_Encrypt_IT()*
- *HAL_CRYP_AESCBC_Encrypt_IT()*
- *HAL_CRYP_AESCTR_Encrypt_IT()*
- *HAL_CRYP_AESECB_Decrypt_IT()*
- *HAL_CRYP_AESCBC_Decrypt_IT()*
- *HAL_CRYP_AESCTR_Decrypt_IT()*
- *HAL_CRYP_AESECB_Encrypt_DMA()*
- *HAL_CRYP_AESCBC_Encrypt_DMA()*
- *HAL_CRYP_AESCTR_Encrypt_DMA()*
- *HAL_CRYP_AESECB_Decrypt_DMA()*
- *HAL_CRYP_AESCBC_Decrypt_DMA()*
- *HAL_CRYP_AESCTR_Decrypt_DMA()*

12.2.4 Callback functions

This section provides Interruption and DMA callback functions:

- DMA Input data transfer complete
- DMA Output data transfer complete
- DMA or Interrupt error

This section contains the following APIs:

- *HAL_CRYP_ErrorCallback()*
- *HAL_CRYP_InCpltCallback()*
- *HAL_CRYP_OutCpltCallback()*

12.2.5 AES IRQ handler management

This section provides AES IRQ handler function.

This section contains the following APIs:

- *HAL_CRYP_IRQHandler()*

12.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_CRYP_GetState()*
- *HAL_CRYP_GetError()*

12.2.7 Detailed description of functions

HAL_CRYP_Init

Function name	**HAL_StatusTypeDef HAL_CRYP_Init (CRYP_HandleTypeDef * hcryp)**
Function description	Initialize the CRYP according to the specified parameters in the CRYP_InitTypeDef and initialize the associated handle.
Return values	• **HAL:** status
Notes	• Specific care must be taken to format the key and the Initialization Vector IV stored in the MCU memory before calling HAL_CRYP_Init(). Refer to explanations hereabove.

HAL_CRYP_DeInit

Function name	**HAL_StatusTypeDef HAL_CRYP_DeInit (CRYP_HandleTypeDef * hcryp)**
Function description	DeInitialize the CRYP peripheral.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **HAL:** status

HAL_CRYP_MspInit

Function name	**void HAL_CRYP_MspInit (CRYP_HandleTypeDef * hcryp)**
Function description	Initialize the CRYP MSP.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None:**

HAL_CRYP_MspDeInit

Function name	**void HAL_CRYP_MspDeInit (CRYP_HandleTypeDef * hcryp)**
Function description	DeInitialize CRYP MSP.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None:**

HAL_CRYP_AESECB_Encrypt

Function name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function description	Encrypt pPlainData in AES ECB encryption mode.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer

	• **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pCypherData:** Pointer to the cyphertext buffer • **Timeout:** Specify Timeout value
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES() API instead (usage recommended).

HAL_CRYP_AESECB_Decrypt

Function name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
Function description	Decrypt pCypherData in AES ECB decryption mode with key derivation, the decyphered data are available in pPlainData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pPlainData:** Pointer to the plaintext buffer • **Timeout:** Specify Timeout value
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES() API instead (usage recommended).

HAL_CRYP_AESCBC_Encrypt

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function description	Encrypt pPlainData in AES CBC encryption mode with key derivation.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pCypherData:** Pointer to the cyphertext buffer • **Timeout:** Specify Timeout value
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES() API instead (usage recommended).

HAL_CRYP_AESCBC_Decrypt

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
Function description	Decrypt pCypherData in AES ECB decryption mode with key derivation, the decyphered data are available in pPlainData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pPlainData:** Pointer to the plaintext buffer • **Timeout:** Specify Timeout value
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES() API instead (usage recommended).

HAL_CRYP_AESCTR_Encrypt

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function description	Encrypt pPlainData in AES CTR encryption mode.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pCypherData:** Pointer to the cyphertext buffer • **Timeout:** Specify Timeout value
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES() API instead (usage recommended).

HAL_CRYP_AESCTR_Decrypt

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
Function description	Decrypt pCypherData in AES CTR decryption mode, the decyphered data are available in pPlainData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16.

- **pPlainData:** Pointer to the plaintext buffer
- **Timeout:** Specify Timeout value

Return values
- **HAL:** status

Notes
- This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES() API instead (usage recommended).

HAL_CRYP_AESECB_Encrypt_IT

Function name
HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)

Function description
Encrypt pPlainData in AES ECB encryption mode using Interrupt, the cypher data are available in pCypherData.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pPlainData:** Pointer to the plaintext buffer
- **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16.
- **pCypherData:** Pointer to the cyphertext buffer

Return values
- **HAL:** status

Notes
- This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_IT() API instead (usage recommended).

HAL_CRYP_AESCBC_Encrypt_IT

Function name
HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)

Function description
Encrypt pPlainData in AES CBC encryption mode using Interrupt, the cypher data are available in pCypherData.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pPlainData:** Pointer to the plaintext buffer
- **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16.
- **pCypherData:** Pointer to the cyphertext buffer

Return values
- **HAL:** status

Notes
- This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_IT() API instead (usage recommended).

HAL_CRYP_AESCTR_Encrypt_IT

Function name
HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)

Function description	Encrypt pPlainData in AES CTR encryption mode using Interrupt, the cypher data are available in pCypherData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pCypherData:** Pointer to the cyphertext buffer
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_IT() API instead (usage recommended).

HAL_CRYP_AESECB_Decrypt_IT

Function name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Decrypt pCypherData in AES ECB decryption mode using Interrupt, the decyphered data are available in pPlainData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pPlainData:** Pointer to the plaintext buffer.
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_IT() API instead (usage recommended).

HAL_CRYP_AESCTR_Decrypt_IT

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Decrypt pCypherData in AES CTR decryption mode using Interrupt, the decyphered data are available in pPlainData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pPlainData:** Pointer to the plaintext buffer
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_IT() API instead (usage recommended).

HAL_CRYP_AESCBC_Decrypt_IT

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Decrypt pCypherData in AES CBC decryption mode using Interrupt, the decyphered data are available in pPlainData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pPlainData:** Pointer to the plaintext buffer
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_IT() API instead (usage recommended).

HAL_CRYP_AESECB_Encrypt_DMA

Function name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function description	Encrypt pPlainData in AES ECB encryption mode using DMA, the cypher data are available in pCypherData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pCypherData:** Pointer to the cyphertext buffer
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_DMA() API instead (usage recommended). • pPlainData and pCypherData buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the IP.

HAL_CRYP_AESECB_Decrypt_DMA

Function name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Decrypt pCypherData in AES ECB decryption mode using DMA, the decyphered data are available in pPlainData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer

- **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16.
- **pPlainData:** Pointer to the plaintext buffer

Return values
- **HAL:** status

Notes
- This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_DMA() API instead (usage recommended).
- pPlainData and pCypherData buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the IP.

HAL_CRYP_AESCBC_Encrypt_DMA

Function name
HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)

Function description
Encrypt pPlainData in AES CBC encryption mode using DMA, the cypher data are available in pCypherData.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pPlainData:** Pointer to the plaintext buffer
- **Size:** Length of the plaintext buffer, must be a multiple of 16.
- **pCypherData:** Pointer to the cyphertext buffer

Return values
- **HAL:** status

Notes
- This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_DMA() API instead (usage recommended).
- pPlainData and pCypherData buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the IP.

HAL_CRYP_AESCBC_Decrypt_DMA

Function name
HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)

Function description
Decrypt pCypherData in AES CBC decryption mode using DMA, the decyphered data are available in pPlainData.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pCypherData:** Pointer to the cyphertext buffer
- **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16.
- **pPlainData:** Pointer to the plaintext buffer

Return values
- **HAL:** status

Notes
- This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_DMA() API instead (usage recommended).

- pPlainData and pCypherData buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the IP.

HAL_CRYP_AESCTR_Encrypt_DMA

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function description	Encrypt pPlainData in AES CTR encryption mode using DMA, the cypher data are available in pCypherData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData:** Pointer to the plaintext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pCypherData:** Pointer to the cyphertext buffer.
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_DMA() API instead (usage recommended). • pPlainData and pCypherData buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the IP.

HAL_CRYP_AESCTR_Decrypt_DMA

Function name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function description	Decrypt pCypherData in AES CTR decryption mode using DMA, the decyphered data are available in pPlainData.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData:** Pointer to the cyphertext buffer • **Size:** Length of the plaintext buffer in bytes, must be a multiple of 16. • **pPlainData:** Pointer to the plaintext buffer
Return values	• **HAL:** status
Notes	• This API is provided only to maintain compatibility with legacy software. Users should directly resort to generic HAL_CRYPEx_AES_DMA() API instead (usage recommended). • pPlainData and pCypherData buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the IP.

HAL_CRYP_InCpltCallback

Function name	**void HAL_CRYP_InCpltCallback (CRYP_HandleTypeDef * hcryp)**

Function description	Input DMA transfer complete callback.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None:**

HAL_CRYP_OutCpltCallback

Function name	**void HAL_CRYP_OutCpltCallback (CRYP_HandleTypeDef * hcryp)**
Function description	Output DMA transfer complete callback.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None:**

HAL_CRYP_ErrorCallback

Function name	**void HAL_CRYP_ErrorCallback (CRYP_HandleTypeDef * hcryp)**
Function description	CRYP error callback.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None:**

HAL_CRYP_IRQHandler

Function name	**void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * hcryp)**
Function description	Handle AES interrupt request.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None:**

HAL_CRYP_GetState

Function name	**HAL_CRYP_STATETypeDef HAL_CRYP_GetState (CRYP_HandleTypeDef * hcryp)**
Function description	Return the CRYP handle state.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **HAL:** state

HAL_CRYP_GetError

Function name	**uint32_t HAL_CRYP_GetError (CRYP_HandleTypeDef * hcryp)**
Function description	Return the CRYP peripheral error.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that

		contains the configuration information for CRYP module
Return values	•	**Error:** bit-map
Notes	•	The returned error is a bit-map combination of possible errors

12.3 CRYP Firmware driver defines

12.3.1 CRYP

AES chaining mode

CRYP_CHAINMODE_AES_ECB	Electronic codebook chaining algorithm
CRYP_CHAINMODE_AES_CBC	Cipher block chaining algorithm
CRYP_CHAINMODE_AES_CTR	Counter mode chaining algorithm
CRYP_CHAINMODE_AES_GCM_GMAC	Galois counter mode - Galois message authentication code
CRYP_CHAINMODE_AES_CCM	Counter with Cipher Mode

AES operating mode

CRYP_ALGOMODE_ENCRYPT	Encryption mode
CRYP_ALGOMODE_KEYDERIVATION	Key derivation mode
CRYP_ALGOMODE_DECRYPT	Decryption
CRYP_ALGOMODE_KEYDERIVATION_DECRYPT	Key derivation and decryption
CRYP_ALGOMODE_TAG_GENERATION	GMAC or CMAC (when applicable) authentication tag generation

AES Enable state

CRYP_AES_DISABLE	Disable AES
CRYP_AES_ENABLE	Enable AES

AES clearing flags

CRYP_CCF_CLEAR	Computation Complete Flag Clear
CRYP_ERR_CLEAR	Error Flag Clear

AES Data Type selection

CRYP_DATATYPE_32B	32-bit data type (no swapping)
CRYP_DATATYPE_16B	16-bit data type (half-word swapping)
CRYP_DATATYPE_8B	8-bit data type (byte swapping)
CRYP_DATATYPE_1B	1-bit data type (bit swapping)

DMA Input phase management enable state

CRYP_DMAIN_DISABLE	Disable DMA Input phase management
CRYP_DMAIN_ENABLE	Enable DMA Input phase management

DMA Output phase management enable state

CRYP_DMAOUT_DISABLE	Disable DMA Output phase management
CRYP_DMAOUT_ENABLE	Enable DMA Output phase management

CRYP Exported Macros

__HAL_CRYP_RESET_HANDLE_STATE	**Description:** • Reset CRYP handle state. **Parameters:** • __HANDLE__: specifies the CRYP handle. **Return value:** • None
__HAL_CRYP_ENABLE	**Description:** • Enable the CRYP AES peripheral. **Parameters:** • __HANDLE__: specifies the CRYP handle. **Return value:** • None
__HAL_CRYP_DISABLE	**Description:** • Disable the CRYP AES peripheral. **Parameters:** • __HANDLE__: specifies the CRYP handle. **Return value:** • None
__HAL_CRYP_SET_OPERATINGMODE	**Description:** • Set the algorithm operating mode. **Parameters:** • __HANDLE__: specifies the CRYP handle. • __OPERATING_MODE__: specifies the operating mode This parameter can be one of the following values: – CRYP_ALGOMODE_ENCRYPT encryption – CRYP_ALGOMODE_KEYDERIVATION key derivation – CRYP_ALGOMODE_DECRYPT decryption – CRYP_ALGOMODE_KEYDERIVATION_DECRYPT key derivation and decryption **Return value:** • None
__HAL_CRYP_SET_CHAININGMODE	**Description:** • Set the algorithm chaining mode. **Parameters:** • __HANDLE__: specifies the CRYP handle. • __CHAINING_MODE__: specifies the chaining mode This parameter can be one of the following

values:
- – CRYP_CHAINMODE_AES_ECB Electronic CodeBook
- – CRYP_CHAINMODE_AES_CBC Cipher Block Chaining
- – CRYP_CHAINMODE_AES_CTR CounTeR mode
- – CRYP_CHAINMODE_AES_GCM_GMAC Galois Counter Mode or Galois Message Authentication Code
- – CRYP_CHAINMODE_AES_CMAC Cipher Message Authentication Code (or Counter with Cipher Mode when applicable)

Return value:

- None

__HAL_CRYP_GET_FLAG

Description:

- Check whether the specified CRYP status flag is set or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - – CRYP_FLAG_BUSY GCM process suspension forbidden
  - – CRYP_IT_WRERR Write Error
  - – CRYP_IT_RDERR Read Error
  - – CRYP_IT_CCF Computation Complete

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_CRYP_CLEAR_FLAG

Description:

- Clear the CRYP pending status flag.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - – CRYP_ERR_CLEAR Read (RDERR) or Write Error (WRERR) Flag Clear
  - – CRYP_CCF_CLEAR Computation Complete Flag (CCF) Clear

Return value:

- None

__HAL_CRYP_GET_IT_SOURCE

Description:

- Check whether the specified CRYP interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP interrupt source to check This parameter can be one of the following values:
  - CRYP_IT_ERRIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt

Return value:

- State: of interruption (TRUE or FALSE).

__HAL_CRYP_GET_IT

Description:

- Check whether the specified CRYP interrupt is set or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: specifies the interrupt to check. This parameter can be one of the following values:
  - CRYP_IT_WRERR Write Error
  - CRYP_IT_RDERR Read Error
  - CRYP_IT_CCF Computation Complete

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_CRYP_CLEAR_IT

Description:

- Clear the CRYP pending interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: specifies the IT to clear. This parameter can be one of the following values:
  - CRYP_ERR_CLEAR Read (RDERR) or Write Error (WRERR) Flag Clear
  - CRYP_CCF_CLEAR Computation Complete Flag (CCF) Clear

Return value:

- None

__HAL_CRYP_ENABLE_IT

Description:

- Enable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values:
  - CRYP_IT_ERRIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete

interrupt

Return value:

- None

__HAL_CRYP_DISABLE_IT	**Description:** • Disable the CRYP interrupt. **Parameters:** • __HANDLE__: specifies the CRYP handle. • __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values: – CRYP_IT_ERRIE Error interrupt (used for RDERR and WRERR) – CRYP_IT_CCFIE Computation Complete interrupt **Return value:** • None

CRYP Exported Types

HAL_CRYP_ERROR_NONE	No error
HAL_CRYP_WRITE_ERROR	Write error
HAL_CRYP_READ_ERROR	Read error
HAL_CRYP_DMA_ERROR	DMA error
HAL_CRYP_BUSY_ERROR	Busy flag error

AES status flags

CRYP_FLAG_BUSY	GCM process suspension forbidden
CRYP_FLAG_WRERR	Write Error
CRYP_FLAG_RDERR	Read error
CRYP_FLAG_CCF	Computation completed

GCM/GMAC and CCM/CMAC (when applicable) processing phase selection

CRYP_GCM_INIT_PHASE	GCM/GMAC (or CCM) init phase
CRYP_GCMCMAC_HEADER_PHASE	GCM/GMAC/CCM/CMAC header phase
CRYP_GCM_PAYLOAD_PHASE	GCM/CCM payload phase
CRYP_GCMCMAC_FINAL_PHASE	GCM/GMAC/CCM/CMAC final phase
CRYP_INIT_PHASE	Init phase
CRYP_HEADER_PHASE	Header phase
CRYP_PAYLOAD_PHASE	Payload phase
CRYP_FINAL_PHASE	Final phase

AES Interrupts flags

CRYP_IT_WRERR	Write Error
CRYP_IT_RDERR	Read Error

CRYP_IT_CCF	Computation completed

Key size selection

CRYP_KEYSIZE_128B	128-bit long key
CRYP_KEYSIZE_256B	256-bit long key

AES decryption key write-up flag

CRYP_KEY_WRITE_ENABLE	Enable decryption key writing
CRYP_KEY_WRITE_DISABLE	Disable decryption key writing

13 HAL CRYP Extension Driver

13.1 CRYPEx Firmware driver API description

13.1.1 Extended callback functions

This section provides callback function:

- Computation completed.

This section contains the following APIs:

- *HAL_CRYPEx_ComputationCpltCallback()*

13.1.2 AES extended processing functions

This section provides functions allowing to:

- Encrypt plaintext or decrypt cipher text using AES algorithm in different chaining modes. Functions are generic (handles ECB, CBC and CTR and all modes) and are only differentiated based on the processing type. Three processing types are available:
  - Polling mode
  - Interrupt mode
  - DMA mode
- Generate and authentication tag in addition to encrypt/decrypt a plain/cipher text using AES algorithm in different chaining modes. Functions are generic (handles GCM, GMAC, CMAC and CCM when applicable) and process only one phase so that steps can be skipped if so required. Functions are only differentiated based on the processing type. Three processing types are available:
  - Polling mode
  - Interrupt mode
  - DMA mode

This section contains the following APIs:

- *HAL_CRYPEx_AES()*
- *HAL_CRYPEx_AES_IT()*
- *HAL_CRYPEx_AES_DMA()*
- *HAL_CRYPEx_AES_Auth()*
- *HAL_CRYPEx_AES_Auth_IT()*
- *HAL_CRYPEx_AES_Auth_DMA()*

13.1.3 AES extended suspension and resumption functions

This section provides functions allowing to:

- save in memory the Initialization Vector, the Key registers, the Control register or the Suspend registers when a process is suspended by a higher priority message
- write back in CRYP hardware block the saved values listed above when the suspended lower priority message processing is resumed.

This section contains the following APIs:

- *HAL_CRYPEx_Read_IVRegisters()*
- *HAL_CRYPEx_Write_IVRegisters()*
- *HAL_CRYPEx_Read_SuspendRegisters()*

- *HAL_CRYPEx_Write_SuspendRegisters()*
- *HAL_CRYPEx_Read_KeyRegisters()*
- *HAL_CRYPEx_Write_KeyRegisters()*
- *HAL_CRYPEx_Read_ControlRegister()*
- *HAL_CRYPEx_Write_ControlRegister()*
- *HAL_CRYPEx_ProcessSuspend()*

13.1.4 Detailed description of functions

HAL_CRYPEx_ComputationCpltCallback

Function name	**void HAL_CRYPEx_ComputationCpltCallback (CRYP_HandleTypeDef * hcryp)**
Function description	Computation completed callbacks.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **None:**

HAL_CRYPEx_AES

Function name	**HAL_StatusTypeDef HAL_CRYPEx_AES (CRYP_HandleTypeDef * hcryp, uint8_t * pInputData, uint16_t Size, uint8_t * pOutputData, uint32_t Timeout)**
Function description	Carry out in polling mode the ciphering or deciphering operation according to hcryp->Init structure fields, all operating modes (encryption, key derivation and/or decryption) and chaining modes ECB, CBC and CTR are managed by this function in polling mode.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pInputData:** Pointer to the plain text in case of encryption or cipher text in case of decryption or key derivation+decryption. Parameter is meaningless in case of key derivation. • **Size:** Length of the input data buffer in bytes, must be a multiple of 16. Parameter is meaningless in case of key derivation. • **pOutputData:** Pointer to the cipher text in case of encryption or plain text in case of decryption/key derivation+decryption, or pointer to the derivative keys in case of key derivation only. • **Timeout:** Specify Timeout value
Return values	• **HAL:** status

HAL_CRYPEx_AES_IT

Function name	**HAL_StatusTypeDef HAL_CRYPEx_AES_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pInputData, uint16_t Size, uint8_t * pOutputData)**
Function description	Carry out in interrupt mode the ciphering or deciphering operation according to hcryp->Init structure fields, all operating modes (encryption, key derivation and/or decryption) and chaining modes ECB, CBC and CTR are managed by this function in interrupt

mode.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInputData:** Pointer to the plain text in case of encryption or cipher text in case of decryption or key derivation+decryption. Parameter is meaningless in case of key derivation.
- **Size:** Length of the input data buffer in bytes, must be a multiple of 16. Parameter is meaningless in case of key derivation.
- **pOutputData:** Pointer to the cipher text in case of encryption or plain text in case of decryption/key derivation+decryption, or pointer to the derivative keys in case of key derivation only.

Return values
- **HAL:** status

HAL_CRYPEx_AES_DMA

Function name
HAL_StatusTypeDef HAL_CRYPEx_AES_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pInputData, uint16_t Size, uint8_t * pOutputData)

Function description
Carry out in DMA mode the ciphering or deciphering operation according to hcryp->Init structure fields.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInputData:** Pointer to the plain text in case of encryption or cipher text in case of decryption or key derivation+decryption.
- **Size:** Length of the input data buffer in bytes, must be a multiple of 16.
- **pOutputData:** Pointer to the cipher text in case of encryption or plain text in case of decryption/key derivation+decryption.

Return values
- **HAL:** status

Notes
- Chaining modes ECB, CBC and CTR are managed by this function in DMA mode.
- Supported operating modes are encryption, decryption and key derivation with decryption.
- No DMA channel is provided for key derivation only and therefore, access to AES_KEYRx registers must be done by software.
- This API is not applicable to key derivation only; for such a mode, access to AES_KEYRx registers must be done by software thru HAL_CRYPEx_AES() or HAL_CRYPEx_AES_IT() APIs.
- pInputData and pOutputData buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the IP.

HAL_CRYPEx_AES_Auth

Function name
HAL_StatusTypeDef HAL_CRYPEx_AES_Auth (CRYP_HandleTypeDef * hcryp, uint8_t * pInputData, uint64_t Size, uint8_t * pOutputData, uint32_t Timeout)

Function description
Carry out in polling mode the authentication tag generation as well as the ciphering or deciphering operation according to hcryp->Init

structure fields.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInputData:**
  - pointer to payload data in GCM or CCM payload phase,
  - pointer to B0 block in CMAC header phase,
  - pointer to C block in CMAC final phase.
  - Parameter is meaningless in case of GCM/GMAC/CCM init, header and final phases.
- **Size:**
  - length of the input payload data buffer in bytes in GCM or CCM payload phase,
  - length of B0 block (in bytes) in CMAC header phase,
  - length of C block (in bytes) in CMAC final phase.
  - Parameter is meaningless in case of GCM/GMAC/CCM init and header phases.
  - Parameter is meaningless in case of CCM final phase.
  - Parameter is message length in bytes in case of GCM final phase.
  - Parameter must be set to zero in case of GMAC final phase.
- **pOutputData:**
  - pointer to plain or cipher text in GCM/CCM payload phase,
  - pointer to authentication tag in GCM/GMAC/CCM/CMAC final phase.
  - Parameter is meaningless in case of GCM/GMAC/CCM init and header phases.
  - Parameter is meaningless in case of CMAC header phase.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

Notes

- Supported operating modes are encryption and decryption, supported chaining modes are GCM, GMAC, CMAC and CCM when the latter is applicable.
- Phases are singly processed according to hcryp->Init.GCMCMACPhase so that steps in these specific chaining modes can be skipped by the user if so required.

HAL_CRYPEx_AES_Auth_IT

Function name

HAL_StatusTypeDef HAL_CRYPEx_AES_Auth_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pInputData, uint64_t Size, uint8_t * pOutputData)

Function description

Carry out in interrupt mode the authentication tag generation as well as the ciphering or deciphering operation according to hcryp->Init structure fields.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInputData:**
  - pointer to payload data in GCM or CCM payload phase,

  - pointer to B0 block in CMAC header phase,
  - pointer to C block in CMAC final phase.
  - Parameter is meaningless in case of GCM/GMAC/CCM init, header and final phases.
- **Size:**
  - length of the input payload data buffer in bytes in GCM or CCM payload phase,
  - length of B0 block (in bytes) in CMAC header phase,
  - length of C block (in bytes) in CMAC final phase.
  - Parameter is meaningless in case of GCM/GMAC/CCM init and header phases.
  - Parameter is meaningless in case of CCM final phase.
  - Parameter is message length in bytes in case of GCM final phase.
  - Parameter must be set to zero in case of GMAC final phase.
- **pOutputData:**
  - pointer to plain or cipher text in GCM/CCM payload phase,
  - pointer to authentication tag in GCM/GMAC/CCM/CMAC final phase.
  - Parameter is meaningless in case of GCM/GMAC/CCM init and header phases.
  - Parameter is meaningless in case of CMAC header phase.

Return values
- **HAL:** status

Notes
- Supported operating modes are encryption and decryption, supported chaining modes are GCM, GMAC and CMAC.
- Phases are singly processed according to hcryp->Init.GCMCMACPhase so that steps in these specific chaining modes can be skipped by the user if so required.

HAL_CRYPEx_AES_Auth_DMA

Function name
HAL_StatusTypeDef HAL_CRYPEx_AES_Auth_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pInputData, uint64_t Size, uint8_t * pOutputData)

Function description
Carry out in DMA mode the authentication tag generation as well as the ciphering or deciphering operation according to hcryp->Init structure fields.

Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInputData:**
  - pointer to payload data in GCM or CCM payload phase,
  - pointer to B0 block in CMAC header phase,
  - pointer to C block in CMAC final phase.
  - Parameter is meaningless in case of GCM/GMAC/CCM init, header and final phases.
- **Size:**
  - length of the input payload data buffer in bytes in GCM or CCM payload phase,
  - length of B0 block (in bytes) in CMAC header phase,

  - length of C block (in bytes) in CMAC final phase.
  - Parameter is meaningless in case of GCM/GMAC/CCM init and header phases.
  - Parameter is meaningless in case of CCM final phase.
  - Parameter is message length in bytes in case of GCM final phase.
  - Parameter must be set to zero in case of GMAC final phase.
- **pOutputData:**
  - pointer to plain or cipher text in GCM/CCM payload phase,
  - pointer to authentication tag in GCM/GMAC/CCM/CMAC final phase.
  - Parameter is meaningless in case of GCM/GMAC/CCM init and header phases.
  - Parameter is meaningless in case of CMAC header phase.

Return values	• **HAL:** status
Notes	• Supported operating modes are encryption and decryption, supported chaining modes are GCM, GMAC and CMAC. • Phases are singly processed according to hcryp->Init.GCMCMACPhase so that steps in these specific chaining modes can be skipped by the user if so required. • pInputData and pOutputData buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the IP.

HAL_CRYPEx_Read_IVRegisters

Function name	**void HAL_CRYPEx_Read_IVRegisters (CRYP_HandleTypeDef * hcryp, uint8_t * Output)**
Function description	In case of message processing suspension, read the Initialization Vector.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module. • **Output:** Pointer to the buffer containing the saved Initialization Vector.
Return values	• **None:**
Notes	• This value has to be stored for reuse by writing the AES_IVRx registers as soon as the interrupted processing has to be resumed. Applicable to all chaining modes. • AES must be disabled when reading or resetting the IV values.

HAL_CRYPEx_Write_IVRegisters

Function name	**void HAL_CRYPEx_Write_IVRegisters (CRYP_HandleTypeDef * hcryp, uint8_t * Input)**
Function description	In case of message processing resumption, rewrite the Initialization Vector in the AES_IVRx registers.

Parameters		• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module. • **Input:** Pointer to the buffer containing the saved Initialization Vector to write back in the CRYP hardware block.
Return values		• **None:**
Notes		• Applicable to all chaining modes. • AES must be disabled when reading or resetting the IV values.

HAL_CRYPEx_Read_SuspendRegisters

Function name	**void HAL_CRYPEx_Read_SuspendRegisters (CRYP_HandleTypeDef * hcryp, uint8_t * Output)**
Function description	In case of message GCM/GMAC (CCM/CMAC when applicable) processing suspension, read the Suspend Registers.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module. • **Output:** Pointer to the buffer containing the saved Suspend Registers.
Return values	• **None:**
Notes	• These values have to be stored for reuse by writing back the AES_SUSPxR registers as soon as the interrupted processing has to be resumed.

HAL_CRYPEx_Write_SuspendRegisters

Function name	**void HAL_CRYPEx_Write_SuspendRegisters (CRYP_HandleTypeDef * hcryp, uint8_t * Input)**
Function description	In case of message GCM/GMAC (CCM/CMAC when applicable) processing resumption, rewrite the Suspend Registers in the AES_SUSPxR registers.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module. • **Input:** Pointer to the buffer containing the saved suspend registers to write back in the CRYP hardware block.
Return values	• **None:**

HAL_CRYPEx_Read_KeyRegisters

Function name	**void HAL_CRYPEx_Read_KeyRegisters (CRYP_HandleTypeDef * hcryp, uint8_t * Output, uint32_t KeySize)**
Function description	In case of message GCM/GMAC (CCM/CMAC when applicable) processing suspension, read the Key Registers.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module. • **Output:** Pointer to the buffer containing the saved Key Registers.

	• **KeySize:** Indicates the key size (128 or 256 bits).
Return values	• **None:**
Notes	• These values have to be stored for reuse by writing back the AES_KEYRx registers as soon as the interrupted processing has to be resumed.

HAL_CRYPEx_Write_KeyRegisters

Function name	**void HAL_CRYPEx_Write_KeyRegisters (CRYP_HandleTypeDef * hcryp, uint8_t * Input, uint32_t KeySize)**
Function description	In case of message GCM/GMAC (CCM/CMAC when applicable) processing resumption, rewrite the Key Registers in the AES_KEYRx registers.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module. • **Input:** Pointer to the buffer containing the saved key registers to write back in the CRYP hardware block. • **KeySize:** Indicates the key size (128 or 256 bits)
Return values	• **None:**

HAL_CRYPEx_Read_ControlRegister

Function name	**void HAL_CRYPEx_Read_ControlRegister (CRYP_HandleTypeDef * hcryp, uint8_t * Output)**
Function description	In case of message GCM/GMAC (CCM/CMAC when applicable) processing suspension, read the Control Register.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module. • **Output:** Pointer to the buffer containing the saved Control Register.
Return values	• **None:**
Notes	• This values has to be stored for reuse by writing back the AES_CR register as soon as the interrupted processing has to be resumed.

HAL_CRYPEx_Write_ControlRegister

Function name	**void HAL_CRYPEx_Write_ControlRegister (CRYP_HandleTypeDef * hcryp, uint8_t * Input)**
Function description	In case of message GCM/GMAC (CCM/CMAC when applicable) processing resumption, rewrite the Control Registers in the AES_CR register.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module. • **Input:** Pointer to the buffer containing the saved Control Register to write back in the CRYP hardware block.

Return values	• **None:**

HAL_CRYPEx_ProcessSuspend

Function name	**void HAL_CRYPEx_ProcessSuspend (CRYP_HandleTypeDef * hcryp)**
Function description	Request CRYP processing suspension when in polling or interruption mode.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.
Return values	• **None:**
Notes	• Set the handle field SuspendRequest to the appropriate value so that the on-going CRYP processing is suspended as soon as the required conditions are met. • It is advised not to suspend the CRYP processing when the DMA controller is managing the data transfer

CRYP_AES_Auth_IT

Function name	**HAL_StatusTypeDef CRYP_AES_Auth_IT (CRYP_HandleTypeDef * hcryp)**
Function description	Handle CRYP block input/output data handling under interruption for GCM, GMAC, CCM or CMAC chaining modes.
Parameters	• **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• **HAL:** status
Notes	• The function is called under interruption only, once interruptions have been enabled by HAL_CRYPEx_AES_Auth_IT().

14 HAL DAC Generic Driver

14.1 DAC Firmware driver registers structures

14.1.1 DAC_HandleTypeDef

Data Fields

- ***DAC_TypeDef * Instance***
- ***__IO HAL_DAC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***DMA_HandleTypeDef * DMA_Handle1***
- ***DMA_HandleTypeDef * DMA_Handle2***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***DAC_TypeDef* DAC_HandleTypeDef::Instance***
  Register base address
- ***__IO HAL_DAC_StateTypeDef DAC_HandleTypeDef::State***
  DAC communication state
- ***HAL_LockTypeDef DAC_HandleTypeDef::Lock***
  DAC locking object
- ***DMA_HandleTypeDef* DAC_HandleTypeDef::DMA_Handle1***
  Pointer DMA handler for channel 1
- ***DMA_HandleTypeDef* DAC_HandleTypeDef::DMA_Handle2***
  Pointer DMA handler for channel 2
- ***__IO uint32_t DAC_HandleTypeDef::ErrorCode***
  DAC Error code

14.1.2 DAC_SampleAndHoldConfTypeDef

Data Fields

- ***uint32_t DAC_SampleTime***
- ***uint32_t DAC_HoldTime***
- ***uint32_t DAC_RefreshTime***

Field Documentation

- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_SampleTime***
  Specifies the Sample time for the selected channel. This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_HoldTime***
  Specifies the hold time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_RefreshTime***
  Specifies the refresh time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 255

14.1.3 DAC_ChannelConfTypeDef

Data Fields

- ***uint32_t DAC_HighFrequency***
- ***uint32_t DAC_SampleAndHold***
- ***uint32_t DAC_Trigger***
- ***uint32_t DAC_OutputBuffer***
- ***uint32_t DAC_ConnectOnChipPeripheral***
- ***uint32_t DAC_UserTrimming***
- ***uint32_t DAC_TrimmingValue***
- ***DAC_SampleAndHoldConfTypeDef DAC_SampleAndHoldConfig***

Field Documentation

- ***uint32_t DAC_ChannelConfTypeDef::DAC_HighFrequency***
  Specifies the frequency interface mode This parameter can be a value of ***DAC_HighFrequency***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_SampleAndHold***
  Specifies whether the DAC mode. This parameter can be a value of ***DAC_SampleAndHold***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_Trigger***
  Specifies the external trigger for the selected DAC channel. This parameter can be a value of ***DAC_trigger_selection***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_OutputBuffer***
  Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of ***DAC_output_buffer***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_ConnectOnChipPeripheral***
  Specifies whether the DAC output is connected or not to on chip peripheral . This parameter can be a value of ***DAC_ConnectOnChipPeripheral***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_UserTrimming***
  Specifies the trimming mode This parameter must be a value of ***DAC_UserTrimming*** DAC_UserTrimming is either factory or user trimming
- ***uint32_t DAC_ChannelConfTypeDef::DAC_TrimmingValue***
  Specifies the offset trimming value i.e. when DAC_SampleAndHold is DAC_TRIMMING_USER. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***DAC_SampleAndHoldConfTypeDef DAC_ChannelConfTypeDef::DAC_SampleAndHoldConfig***
  Sample and Hold settings

14.2 DAC Firmware driver API description

14.2.1 DAC Peripheral features

DAC Channels

STM32L4 devices integrate one or two 12-bit Digital Analog Converters (i.e. one or 2 channel(s)) 1 channel: STM32L451xx STM32L452xx STM32L462xx 2 channels: STM32L431xx STM32L432xx STM32L433xx STM32L442xx STM32L443xx STM32L471xx STM32L475xx STM32L476xx STM32L485xx STM32L486xx STM32L496xx STM32L4A6xx STM32L4R5xx STM32L4R7xx STM32L4R9xx STM32L4S5xx STM32L4S7xx STM32L4S9xx When 2 channels are available, the 2 converters (i.e. channel1 & channel2) can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC_OUT1 (PA4) as output or connected to on-chip peripherals (ex. OPAMPs, comparators).
2. Whenever present, DAC channel2 with DAC_OUT2 (PA5) as output or connected to on-chip peripherals (ex. OPAMPs, comparators).

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_TRIGGER_NONE and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_PIN_9) using DAC_TRIGGER_EXT_IT9. The used pin (GPIOx_PIN_9) must be configured in input mode.
2. Timers TRGO: TIM2, TIM3, TIM4, TIM5, TIM6 and TIM7 (DAC_TRIGGER_T2_TRGO, DAC_TRIGGER_T3_TRGO...)
3. Software using DAC_TRIGGER_SOFTWARE

DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;

Refer to the device datasheet for more details about output impedance value with and without output buffer.

DAC connect feature

Each DAC channel can be connected internally. To connect, use sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_ENABLE;

GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel2 on PA5 is configured in Analog and disabled). Channel1 may disturb channel2 as coupling effect. Note that there is no coupling on channel2 as soon as channel2 is turned on. Coupling on adjacent channel could be avoided as follows: when unused PA5 is configured as INPUT PULL-UP or DOWN. PA5 is configured in ANALOG just before it is turned on.

DAC Sample and Hold feature

For each converter, 2 modes are supported: normal mode and "sample and hold" mode (i.e. low power mode). In the sample and hold mode, the DAC core converts data, then holds the converted voltage on a capacitor. When not converting, the DAC cores and buffer are completely turned off between samples and the DAC output is tri-stated, therefore reducing the overall power consumption. A new stabilization period is needed before each new conversion. The sample and hold allow setting internal or external voltage @ low power consumption cost (output value can be at any given rate either by CPU or DMA). The Sample and hold block and registers uses either LSI & run in several power modes: run mode, sleep mode, low power run, low power sleep mode & stop1 mode. Low power stop1 mode allows only static conversion. To enable Sample and Hold mode Enable LSI using HAL_RCC_OscConfig with RCC_OSCILLATORTYPE_LSI & RCC_LSI_ON parameters. Use DAC_InitStructure.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_ENABLE; &

DAC_ChannelConfTypeDef.DAC_SampleAndHoldConfig.DAC_SampleTime, DAC_HoldTime & DAC_RefreshTime;

DAC calibration feature

1. The 2 converters (channel1 & channel2) provide calibration capabilities.
   – Calibration aims at correcting some offset of output buffer.
   – The DAC uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
   – The user defined settings can be figured out using self calibration handled by HAL_DACEx_SelfCalibrate.
   – HAL_DACEx_SelfCalibrate:
     – Runs automatically the calibration.
     – Enables the user trimming mode
     – Updates a structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)

DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave
2. Triangle wave

DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC_ALIGN_8B_R
2. 12-bit left alignment using DAC_ALIGN_12B_L
3. 12-bit right alignment using DAC_ALIGN_12B_R

DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation:

DAC_OUTx = VREF+ * DOR / 4095

- with DOR is the Data Output Register

VEF+ is the input voltage reference (refer to the device datasheet)

e.g. To set DAC_OUT1 to 0.7V, use

- Assuming that VREF+ = 3.3V, DAC_OUT1 = (3.3 * 868) / 4095 = 0.7V

DMA requests

A DMA1 request can be generated when an external trigger (but not a software trigger) occurs if DMA1 requests are enabled using HAL_DAC_Start_DMA(). DMA requests are mapped as following:

1. DAC channel1: mapped either on
   – DMA1 request 6 channel3
   – or DMA2 request channel4 which must be already configured
2. DAC channel2 (whenever present): mapped either on
   – DMA1 request 5 channel4
   – or DMA2 request 3 channel5 which must be already configured

For Dual mode and specific signal (Triangle and noise) generation please refer to Extended Features Driver description

14.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL_DAC_Init()
- Configure DAC_OUTx (DAC_OUT1: PA4, DAC_OUT2: PA5) in analog mode.
- Configure the DAC channel using HAL_DAC_ConfigChannel() function.
- Enable the DAC channel using HAL_DAC_Start() or HAL_DAC_Start_DMA() functions.

Calibration mode IO operation

- Retrieve the factory trimming (calibration settings) using HAL_DACEx_GetTrimOffset()
- Run the calibration using HAL_DACEx_SelfCalibrate()
- Update the trimming while DAC running using HAL_DACEx_SetUserTrimming()

Polling mode IO operation

- Start the DAC peripheral using HAL_DAC_Start()
- To read the DAC last data output value, use the HAL_DAC_GetValue() function.
- Stop the DAC peripheral using HAL_DAC_Stop()

DMA mode IO operation

- Start the DAC peripheral using HAL_DAC_Start_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion
- At the middle of data transfer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- In case of transfer Error, HAL_DAC_ErrorCallbackCh1() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1
- In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() and add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1()
- Stop the DAC peripheral using HAL_DAC_Stop_DMA()

DAC HAL driver macros list

Below the list of most used macros in DAC HAL driver.

- __HAL_DAC_ENABLE: Enable the DAC peripheral
- __HAL_DAC_DISABLE: Disable the DAC peripheral

- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status

You can refer to the DAC HAL driver header file for more useful macros

14.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DAC.
- De-initialize the DAC.

This section contains the following APIs:

- ***HAL_DAC_Init()***
- ***HAL_DAC_DeInit()***
- ***HAL_DAC_MspInit()***
- ***HAL_DAC_MspDeInit()***

14.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.

This section contains the following APIs:

- ***HAL_DAC_Start()***
- ***HAL_DAC_Stop()***
- ***HAL_DAC_Start_DMA()***
- ***HAL_DAC_Stop_DMA()***
- ***HAL_DAC_IRQHandler()***
- ***HAL_DAC_SetValue()***
- ***HAL_DAC_ConvCpltCallbackCh1()***
- ***HAL_DAC_ConvHalfCpltCallbackCh1()***
- ***HAL_DAC_ErrorCallbackCh1()***
- ***HAL_DAC_DMAUnderrunCallbackCh1()***

14.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- ***HAL_DAC_GetValue()***
- ***HAL_DAC_ConfigChannel()***

14.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.

This section contains the following APIs:

- *HAL_DAC_GetState()*
- *HAL_DAC_GetError()*

14.2.7 Detailed description of functions

HAL_DAC_Init

Function name	**HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)**
Function description	Initialize the DAC peripheral according to the specified parameters in the DAC_InitStruct and initialize the associated handle.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **HAL:** status

HAL_DAC_DeInit

Function name	**HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)**
Function description	Deinitialize the DAC peripheral registers to their default reset values.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **HAL:** status

HAL_DAC_MspInit

Function name	**void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)**
Function description	Initialize the DAC MSP.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DAC_MspDeInit

Function name	**void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)**
Function description	DeInitialize the DAC MSP.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DAC_Start

Function name	**HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function description	Enables DAC and starts conversion of channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected (when supported)
Return values	• **HAL:** status

HAL_DAC_Stop

Function name	**HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function description	Disables DAC and stop conversion of channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected
Return values	• **HAL:** status

HAL_DAC_Start_DMA

Function name	**HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)**
Function description	Enables DAC and starts conversion of channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected • **pData:** The destination peripheral Buffer address. • **Length:** The length of data to be transferred from memory to DAC peripheral • **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values: – DAC_ALIGN_8B_R: 8bit right data alignment selected – DAC_ALIGN_12B_L: 12bit left data alignment selected – DAC_ALIGN_12B_R: 12bit right data alignment selected
Return values	• **HAL:** status

HAL_DAC_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function description	Disables DAC and stop conversion of channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected
Return values	• **HAL:** status

HAL_DAC_IRQHandler

Function name	**void HAL_DAC_IRQHandler (DAC_HandleTypeDef * hdac)**
Function description	Handles DAC interrupt request This function uses the interruption of DMA underrun.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DAC_SetValue

Function name	**HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)**
Function description	Set the specified data holding register value for DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected • **Alignment:** Specifies the data alignment. This parameter can be one of the following values: – DAC_ALIGN_8B_R: 8bit right data alignment selected – DAC_ALIGN_12B_L: 12bit left data alignment selected – DAC_ALIGN_12B_R: 12bit right data alignment selected • **Data:** Data to be loaded in the selected data holding register.
Return values	• **HAL:** status

HAL_DAC_ConvCpltCallbackCh1

Function name	**void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)**
Function description	Conversion complete callback in non-blocking mode for Channel1.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that

	contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DAC_ConvHalfCpltCallbackCh1

Function name	**void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)**
Function description	Conversion half DMA transfer callback in non-blocking mode for Channel1.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DAC_ErrorCallbackCh1

Function name	**void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)**
Function description	Error DAC callback for Channel1.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DAC_DMAUnderrunCallbackCh1

Function name	**void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)**
Function description	DMA underrun DAC callback for channel1.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DAC_GetValue

Function name	**uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function description	Returns the last data output value of the selected DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected
Return values	• **The:** selected DAC channel data output value.

HAL_DAC_ConfigChannel

Function name	**HAL_StatusTypeDef HAL_DAC_ConfigChannel**

	(DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)
Function description	Configures the selected DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **sConfig:** DAC configuration structure. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected (Whenever present)
Return values	• **HAL:** status

HAL_DAC_GetState

Function name	**HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)**
Function description	return the DAC handle state
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **HAL:** state

HAL_DAC_GetError

Function name	**uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)**
Function description	Return the DAC error code.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **DAC:** Error Code

14.3 DAC Firmware driver defines

14.3.1 DAC

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC ConnectOnChipPeripheral

DAC_CHIPCONNECT_DISABLE

DAC_CHIPCONNECT_ENABLE

DAC data alignment

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE	No error
HAL_DAC_ERROR_DMAUNDERRUNCH1	DAC channel1 DMA underrun error
HAL_DAC_ERROR_DMAUNDERRUNCH2	DAC channel2 DMA underrun error
HAL_DAC_ERROR_DMA	DMA error
HAL_DAC_ERROR_TIMEOUT	Timeout error

DAC Exported Macros

__HAL_DAC_RESET_HANDLE_STATE

Description:

- Reset DAC handle state.

Parameters:

- __HANDLE__: specifies the DAC handle.

Return value:

- None

__HAL_DAC_ENABLE

Description:

- Enable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __DAC_Channel__: specifies the DAC channel

Return value:

- None

__HAL_DAC_DISABLE

Description:

- Disable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle
- __DAC_Channel__: specifies the DAC channel.

Return value:

- None

DAC_DHR12R1_ALIGNMENT

Description:

- Set DHR12R1 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12R2_ALIGNMENT

Description:

- Set DHR12R2 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12RD_ALIGNMENT

Description:

- Set DHR12RD alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

__HAL_DAC_ENABLE_IT

Description:

- Enable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt. This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1: DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2: DAC channel 2 DMA underrun interrupt

Return value:

- None

__HAL_DAC_DISABLE_IT

Description:

- Disable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt. This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1: DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2: DAC channel 2 DMA underrun interrupt

Return value:

- None

__HAL_DAC_GET_IT_SOURCE

Description:

- Check whether the specified DAC interrupt source is enabled or not.

Parameters:

- __HANDLE__: DAC handle
- __INTERRUPT__: DAC interrupt source to check This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1: DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2: DAC channel 2 DMA underrun interrupt

Return value:

- State: of interruption (SET or RESET)

__HAL_DAC_GET_FLAG

Description:

- Get the selected DAC's flag status.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to get. This parameter can be any combination of the following values:
  - DAC_FLAG_DMAUDR1: DAC channel 1 DMA underrun flag
  - DAC_FLAG_DMAUDR2: DAC channel 2 DMA underrun flag

Return value:

- None

__HAL_DAC_CLEAR_FLAG

Description:

- Clear the DAC's flag.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to clear. This parameter can be any combination of the following values:
  - DAC_FLAG_DMAUDR1: DAC channel 1 DMA underrun flag
  - DAC_FLAG_DMAUDR2: DAC channel 2 DMA underrun flag

Return value:

- None

DAC flags definition

DAC_FLAG_DMAUDR1

DAC_FLAG_DMAUDR2

DAC high frequency interface mode

DAC_HIGH_FREQUENCY_INTERFACE_MODE_DISABLE High frequency interface mode

	disabled
DAC_HIGH_FREQUENCY_INTERFACE_MODE_ABOVE_80MHZ	High frequency interface mode enabled
DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC	High frequency interface mode automatic

DAC IT definition

DAC_IT_DMAUDR1

DAC_IT_DMAUDR2

DAC output buffer

DAC_OUTPUTBUFFER_ENABLE

DAC_OUTPUTBUFFER_DISABLE

DAC power mode

DAC_SAMPLEANDHOLD_DISABLE

DAC_SAMPLEANDHOLD_ENABLE

DAC trigger selection

DAC_TRIGGER_NONE	Conversion is automatic once the DAC_DHRxxxx register has been loaded, and not by external trigger
DAC_TRIGGER_T1_TRGO	TIM1 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T2_TRGO	TIM2 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T4_TRGO	TIM1 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T5_TRGO	TIM5 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T6_TRGO	TIM6 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T7_TRGO	TIM7 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T8_TRGO	TIM8 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T15_TRGO	TIM15 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_LPTIM1_OUT	LPTIM1 OUT TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_LPTIM2_OUT	LPTIM2 OUT TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_EXT_IT9	EXTI Line9 event selected as external conversion trigger for DAC channel
DAC_TRIGGER_SOFTWARE	Conversion started by software trigger for DAC channel

DAC User Trimming

DAC_TRIMMING_FACTORY	Factory trimming
DAC_TRIMMING_USER	User trimming

15 HAL DAC Extension Driver

15.1 DACEx Firmware driver API description

15.1.1 How to use this driver

- When Dual mode is enabled (i.e. DAC Channel1 and Channel2 are used simultaneously): Use HAL_DACEx_DualGetValue() to get digital data to be converted and use HAL_DACEx_DualSetValue() to set digital value to converted simultaneously in Channel 1 and Channel 2.
- Use HAL_DACEx_TriangleWaveGenerate() to generate Triangle signal.
- Use HAL_DACEx_NoiseWaveGenerate() to generate Noise signal.
- HAL_DACEx_SelfCalibrate to calibrate one DAC channel.
- HAL_DACEx_SetUserTrimming to set user trimming value.
- HAL_DACEx_GetTrimOffset to retrieve trimming value (factory setting after reset, user setting if HAL_DACEx_SetUserTrimming have been used at least one time after reset).

15.1.2 Extended features functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion.

This section contains the following APIs:

- ***HAL_DACEx_TriangleWaveGenerate()***
- ***HAL_DACEx_NoiseWaveGenerate()***
- ***HAL_DACEx_DualSetValue()***
- ***HAL_DACEx_ConvCpltCallbackCh2()***
- ***HAL_DACEx_ConvHalfCpltCallbackCh2()***
- ***HAL_DACEx_ErrorCallbackCh2()***
- ***HAL_DACEx_DMAUnderrunCallbackCh2()***
- ***HAL_DACEx_SelfCalibrate()***
- ***HAL_DACEx_SetUserTrimming()***
- ***HAL_DACEx_GetTrimOffset()***

15.1.3 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- ***HAL_DACEx_DualGetValue()***
- ***HAL_DACEx_GetTrimOffset()***

15.1.4 Detailed description of functions

HAL_DACEx_TriangleWaveGenerate

Function name	**HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)**
Function description	Enable or disable the selected DAC channel wave generation.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2 • **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values: – DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1 – DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3 – DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7 – DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15 – DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31 – DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63 – DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127 – DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255 – DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511 – DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023 – DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047 – DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095
Return values	• **HAL:** status

HAL_DACEx_NoiseWaveGenerate

Function name	**HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)**
Function description	Enable or disable the selected DAC channel wave generation.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2

	• **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values: – DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation – DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation – DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation – DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation – DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation – DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation – DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation – DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation – DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation – DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation – DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation – DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation
Return values	• **HAL:** status

HAL_DACEx_DualSetValue

Function name	**HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)**
Function description	Set the specified data holding register value for dual DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Alignment:** Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected • **Data1:** Data for DAC Channel2 to be loaded in the selected data holding register. • **Data2:** Data for DAC Channel1 to be loaded in the selected data holding register.
Return values	• **HAL:** status
Notes	• In dual mode, a unique register access is required to write in both DAC channels at the same time.

HAL_DACEx_ConvCpltCallbackCh2

Function name	**void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)**
Function description	Conversion complete callback in non-blocking mode for Channel2.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DACEx_ConvHalfCpltCallbackCh2

Function name	**void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)**
Function description	Conversion half DMA transfer callback in non-blocking mode for Channel2.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DACEx_ErrorCallbackCh2

Function name	**void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef * hdac)**
Function description	Error DAC callback for Channel2.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DACEx_DMAUnderrunCallbackCh2

Function name	**void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef * hdac)**
Function description	DMA underrun DAC callback for Channel2.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **None:**

HAL_DACEx_SelfCalibrate

Function name	**HAL_StatusTypeDef HAL_DACEx_SelfCalibrate (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)**
Function description	Run the self calibration of one DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **sConfig:** DAC channel configuration structure. • **Channel:** The selected DAC channel. This parameter can be

one of the following values:
- – DAC_CHANNEL_1: DAC Channel1 selected
- – DAC_CHANNEL_2: DAC Channel2 selected

Return values	• **Updates:** DAC_TrimmingValue. , DAC_UserTrimming set to DAC_UserTrimming • **HAL:** status
Notes	• Calibration runs about 7 ms.

HAL_DACEx_SetUserTrimming

Function name	**HAL_StatusTypeDef HAL_DACEx_SetUserTrimming (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel, uint32_t NewTrimmingValue)**
Function description	Set the trimming mode and trimming value (user trimming mode applied).
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **sConfig:** DAC configuration structure updated with new DAC trimming value. • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected • **NewTrimmingValue:** DAC new trimming value
Return values	• **HAL:** status

HAL_DACEx_DualGetValue

Function name	**uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)**
Function description	Return the last data output value of the selected DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **The:** selected DAC channel data output value.

HAL_DACEx_GetTrimOffset

Function name	**uint32_t HAL_DACEx_GetTrimOffset (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function description	Return the DAC trimming value.
Parameters	• **hdac::** DAC handle • **Channel:** The selected DAC channel. This parameter can be one of the following values: – DAC_CHANNEL_1: DAC Channel1 selected – DAC_CHANNEL_2: DAC Channel2 selected
Return values	• **Trimming:** value: range: 0->31

DAC_DMAConvCpltCh2

Function name	**void DAC_DMAConvCpltCh2 (DMA_HandleTypeDef * hdma)**
Function description	DMA conversion complete callback.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• **None:**

DAC_DMAErrorCh2

Function name	**void DAC_DMAErrorCh2 (DMA_HandleTypeDef * hdma)**
Function description	DMA error callback.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• **None:**

DAC_DMAHalfConvCpltCh2

Function name	**void DAC_DMAHalfConvCpltCh2 (DMA_HandleTypeDef * hdma)**
Function description	DMA half transfer complete callback.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• **None:**

15.2 DACEx Firmware driver defines

15.2.1 DACEx

DACEx lfsrunmask triangle amplitude

DAC_LFSRUNMASK_BIT0	Unmask DAC channel LFSR bit0 for noise wave generation
DAC_LFSRUNMASK_BITS1_0	Unmask DAC channel LFSR bit[1:0] for noise wave generation
DAC_LFSRUNMASK_BITS2_0	Unmask DAC channel LFSR bit[2:0] for noise wave generation
DAC_LFSRUNMASK_BITS3_0	Unmask DAC channel LFSR bit[3:0] for noise wave generation
DAC_LFSRUNMASK_BITS4_0	Unmask DAC channel LFSR bit[4:0] for noise wave generation
DAC_LFSRUNMASK_BITS5_0	Unmask DAC channel LFSR bit[5:0] for noise wave generation
DAC_LFSRUNMASK_BITS6_0	Unmask DAC channel LFSR bit[6:0] for noise wave

	generation
DAC_LFSRUNMASK_BITS7_0	Unmask DAC channel LFSR bit[7:0] for noise wave generation
DAC_LFSRUNMASK_BITS8_0	Unmask DAC channel LFSR bit[8:0] for noise wave generation
DAC_LFSRUNMASK_BITS9_0	Unmask DAC channel LFSR bit[9:0] for noise wave generation
DAC_LFSRUNMASK_BITS10_0	Unmask DAC channel LFSR bit[10:0] for noise wave generation
DAC_LFSRUNMASK_BITS11_0	Unmask DAC channel LFSR bit[11:0] for noise wave generation
DAC_TRIANGLEAMPLITUDE_1	Select max triangle amplitude of 1
DAC_TRIANGLEAMPLITUDE_3	Select max triangle amplitude of 3
DAC_TRIANGLEAMPLITUDE_7	Select max triangle amplitude of 7
DAC_TRIANGLEAMPLITUDE_15	Select max triangle amplitude of 15
DAC_TRIANGLEAMPLITUDE_31	Select max triangle amplitude of 31
DAC_TRIANGLEAMPLITUDE_63	Select max triangle amplitude of 63
DAC_TRIANGLEAMPLITUDE_127	Select max triangle amplitude of 127
DAC_TRIANGLEAMPLITUDE_255	Select max triangle amplitude of 255
DAC_TRIANGLEAMPLITUDE_511	Select max triangle amplitude of 511
DAC_TRIANGLEAMPLITUDE_1023	Select max triangle amplitude of 1023
DAC_TRIANGLEAMPLITUDE_2047	Select max triangle amplitude of 2047
DAC_TRIANGLEAMPLITUDE_4095	Select max triangle amplitude of 4095

16 HAL DCMI Generic Driver

16.1 DCMI Firmware driver registers structures

16.1.1 DCMI_CodesInitTypeDef

Data Fields

- ***uint8_t FrameStartCode***
- ***uint8_t LineStartCode***
- ***uint8_t LineEndCode***
- ***uint8_t FrameEndCode***

Field Documentation

- ***uint8_t DCMI_CodesInitTypeDef::FrameStartCode***
  Specifies the code of the frame start delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::LineStartCode***
  Specifies the code of the line start delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::LineEndCode***
  Specifies the code of the line end delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::FrameEndCode***
  Specifies the code of the frame end delimiter.

16.1.2 DCMI_SyncUnmaskTypeDef

Data Fields

- ***uint8_t FrameStartUnmask***
- ***uint8_t LineStartUnmask***
- ***uint8_t LineEndUnmask***
- ***uint8_t FrameEndUnmask***

Field Documentation

- ***uint8_t DCMI_SyncUnmaskTypeDef::FrameStartUnmask***
  Specifies the frame start delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::LineStartUnmask***
  Specifies the line start delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::LineEndUnmask***
  Specifies the line end delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::FrameEndUnmask***
  Specifies the frame end delimiter unmask.

16.1.3 DCMI_InitTypeDef

Data Fields

- ***uint32_t SynchroMode***
- ***uint32_t PCKPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t HSPolarity***
- ***uint32_t CaptureRate***
- ***uint32_t ExtendedDataMode***
- ***DCMI_CodesInitTypeDef SynchroCode***

- ***uint32_t JPEGMode***
- ***uint32_t ByteSelectMode***
- ***uint32_t ByteSelectStart***
- ***uint32_t LineSelectMode***
- ***uint32_t LineSelectStart***

Field Documentation

- ***uint32_t DCMI_InitTypeDef::SynchroMode***
  Specifies the Synchronization Mode: Hardware or Embedded. This parameter can be a value of ***DCMI_Synchronization_Mode***.
- ***uint32_t DCMI_InitTypeDef::PCKPolarity***
  Specifies the Pixel clock polarity: Falling or Rising. This parameter can be a value of ***DCMI_PIXCK_Polarity***.
- ***uint32_t DCMI_InitTypeDef::VSPolarity***
  Specifies the Vertical synchronization polarity: High or Low. This parameter can be a value of ***DCMI_VSYNC_Polarity***.
- ***uint32_t DCMI_InitTypeDef::HSPolarity***
  Specifies the Horizontal synchronization polarity: High or Low. This parameter can be a value of ***DCMI_HSYNC_Polarity***.
- ***uint32_t DCMI_InitTypeDef::CaptureRate***
  Specifies the frequency of frame capture: All, 1/2 or 1/4. This parameter can be a value of ***DCMI_Capture_Rate***.
- ***uint32_t DCMI_InitTypeDef::ExtendedDataMode***
  Specifies the data width: 8-bit, 10-bit, 12-bit or 14-bit. This parameter can be a value of ***DCMI_Extended_Data_Mode***.
- ***DCMI_CodesInitTypeDef DCMI_InitTypeDef::SynchroCode***
  Specifies the frame start delimiter codes.
- ***uint32_t DCMI_InitTypeDef::JPEGMode***
  Enable or Disable the JPEG mode. This parameter can be a value of ***DCMI_JPEG_Mode***.
- ***uint32_t DCMI_InitTypeDef::ByteSelectMode***
  Specifies the data to be captured by the interface. This parameter can be a value of ***DCMI_Byte_Select_Mode***.
- ***uint32_t DCMI_InitTypeDef::ByteSelectStart***
  Specifies if the data to be captured by the interface is even or odd. This parameter can be a value of ***DCMI_Byte_Select_Start***.
- ***uint32_t DCMI_InitTypeDef::LineSelectMode***
  Specifies the data line to be captured by the interface. This parameter can be a value of ***DCMI_Line_Select_Mode***.
- ***uint32_t DCMI_InitTypeDef::LineSelectStart***
  Specifies if the data line to be captured by the interface is even or odd. This parameter can be a value of ***DCMI_Line_Select_Start***.

16.1.4 DCMI_HandleTypeDef

Data Fields

- ***DCMI_TypeDef * Instance***
- ***DCMI_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DCMI_StateTypeDef State***
- ***__IO uint32_t XferCount***
- ***__IO uint32_t XferSize***
- ***uint32_t pBuffPtr***
- ***DMA_HandleTypeDef * DMA_Handle***

- ***DMA_HandleTypeDef * DMAM2M_Handle***
- ***__IO uint32_t ErrorCode***
- ***uint32_t pCircularBuffer***
- ***uint32_t HalfCopyLength***

Field Documentation

- ***DCMI_TypeDef* DCMI_HandleTypeDef::Instance***
  DCMI Register base address
- ***DCMI_InitTypeDef DCMI_HandleTypeDef::Init***
  DCMI init parameters
- ***HAL_LockTypeDef DCMI_HandleTypeDef::Lock***
  DCMI locking object
- ***__IO HAL_DCMI_StateTypeDef DCMI_HandleTypeDef::State***
  DCMI state
- ***__IO uint32_t DCMI_HandleTypeDef::XferCount***
  DMA transfers counter
- ***__IO uint32_t DCMI_HandleTypeDef::XferSize***
  DMA transfer size
- ***uint32_t DCMI_HandleTypeDef::pBuffPtr***
  Pointer to DMA output buffer
- ***DMA_HandleTypeDef* DCMI_HandleTypeDef::DMA_Handle***
  Pointer to DMA handler
- ***DMA_HandleTypeDef* DCMI_HandleTypeDef::DMAM2M_Handle***
  Pointer to DMA handler for memory to memory copy (case picture size > maximum DMA transfer length)
- ***__IO uint32_t DCMI_HandleTypeDef::ErrorCode***
  DCMI Error code
- ***uint32_t DCMI_HandleTypeDef::pCircularBuffer***
  Pointer to intermediate copy buffer (case picture size > maximum DMA transfer length)
- ***uint32_t DCMI_HandleTypeDef::HalfCopyLength***
  Intermediate copies length (case picture size > maximum DMA transfer length)

16.2 DCMI Firmware driver API description

16.2.1 How to use this driver

The sequence below describes how to use this driver to capture an image from a camera module connected to the DCMI Interface. This sequence does not take into account the configuration of the camera module, which should be made before configuring and enabling the DCMI.

1. Program the required configuration through the following parameters: horizontal and vertical polarity, pixel clock polarity, capture rate, synchronization mode, frame delimiter codes, data width, byte and line selection using HAL_DCMI_Init() function.
2. Optionally select JPEG mode; in that case, only the polarity and the capture mode parameters need to be set.
3. Capture mode can be either snapshot or continuous mode.
4. Configure the DMA_Handle to transfer data from DCMI DR register to the destination memory buffer. In snapshot mode, the interface transfers a single frame through DMA. In continuous mode, the DMA must be set in circular mode to ensure a continuous flow of images data samples.
5. Program the transfer configuration through the following parameters: DCMI mode, destination memory buffer address and data length then enable capture using HAL_DCMI_Start_DMA() function.

6. Whether in continuous or snapshot mode, data length parameter must be equal to the frame size.
7. When the frame size is unknown beforehand (e.g. JPEG case), data length must be large enough to ensure the capture of a frame.
8. If the frame size is larger than the maximum DMA transfer length (i.e. 65535),
   - the DMA must be configured in circular mode, either for snapshot or continuous capture mode,
   - during capture, the driver copies the image data samples from DCMI DR register at the end of the final destination buffer used as a work buffer,
   - at each DMA half (respectively complete) transfer interrupt, the first (resp. second) half of the work buffer is copied to the final destination thru a second DMA channel.
   - Parameters of this second DMA channel are contained in the memory to memory DMA handle "DMAM2M_Handle", itself field of the DCMI handle structure.
   - This memory to memory transfer has length half that of the work buffer and is carried out in normal mode (not in circular mode).
9. Optionally, configure and enable the CROP feature to select a rectangular window from the received image using HAL_DCMI_ConfigCrop() and HAL_DCMI_EnableCrop() functions. Use HAL_DCMI_DisableCrop() to disable this feature.
10. The capture can be stopped with HAL_DCMI_Stop() function.
11. To control the DCMI state, use the function HAL_DCMI_GetState().
12. To read the DCMI error code, use the function HAL_DCMI_GetError().

When the frame size is less than the maximum DMA transfer length (i.e. 65535) and when in snapshot mode, user must make sure the FRAME interrupt is disabled. This allows to avoid corner cases where the FRAME interrupt might be triggered before the DMA transfer completion interrupt. In this specific configuration, the driver checks the FRAME capture flag after the DMA transfer end and calls HAL_DCMI_FrameEventCallback() if the flag is set.

DCMI HAL driver macros list

Below the list of most used macros in DCMI HAL driver.

- __HAL_DCMI_ENABLE: Enable the DCMI peripheral.
- __HAL_DCMI_DISABLE: Disable the DCMI peripheral.
- __HAL_DCMI_GET_FLAG: Get the DCMI pending flags.
- __HAL_DCMI_CLEAR_FLAG: Clear the DCMI pending flags.
- __HAL_DCMI_ENABLE_IT: Enable the specified DCMI interrupts.
- __HAL_DCMI_DISABLE_IT: Disable the specified DCMI interrupts.
- __HAL_DCMI_GET_IT_SOURCE: Check whether the specified DCMI interrupt has occurred and that the interruption is enabled at the same time.

16.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DCMI
- De-initialize the DCMI

This section contains the following APIs:

- ***HAL_DCMI_Init()***
- ***HAL_DCMI_DeInit()***
- ***HAL_DCMI_MspInit()***

- ***HAL_DCMI_MspDeInit()***

16.2.3 IO operation functions

This section provides functions allowing to:

- Configure destination address and data length, enable DCMI DMA request and DCMI capture.
- Stop DCMI capture.
- Handle DCMI interrupt request.

A set of callbacks is provided:

- HAL_DCMI_ErrorCallback()
- HAL_DCMI_LineEventCallback()
- HAL_DCMI_VsyncEventCallback()
- HAL_DCMI_FrameEventCallback()

This section contains the following APIs:

- ***HAL_DCMI_Start_DMA()***
- ***HAL_DCMI_Stop()***
- ***HAL_DCMI_Suspend()***
- ***HAL_DCMI_Resume()***
- ***HAL_DCMI_IRQHandler()***
- ***HAL_DCMI_ErrorCallback()***
- ***HAL_DCMI_LineEventCallback()***
- ***HAL_DCMI_VsyncEventCallback()***
- ***HAL_DCMI_FrameEventCallback()***

16.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the crop feature.
- Enable/Disable the crop feature.
- Configure the synchronization delimiters unmasks.
- Enable/Disable user-specified DCMI interrupts.

This section contains the following APIs:

- ***HAL_DCMI_ConfigCrop()***
- ***HAL_DCMI_DisableCrop()***
- ***HAL_DCMI_EnableCrop()***
- ***HAL_DCMI_ConfigSyncUnmask()***

16.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DCMI state.
- Get the specific DCMI error flag.

This section contains the following APIs:

- ***HAL_DCMI_GetState()***
- ***HAL_DCMI_GetError()***

16.2.6 Detailed description of functions

HAL_DCMI_Init

Function name	**HAL_StatusTypeDef HAL_DCMI_Init (DCMI_HandleTypeDef * hdcmi)**
Function description	Initialize the DCMI according to the specified parameters in the DCMI_InitTypeDef and create the associated handle.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **HAL:** status
Notes	• By default, all interruptions are enabled (line end, frame end, overrun, VSYNC and embedded synchronization error interrupts).

HAL_DCMI_DeInit

Function name	**HAL_StatusTypeDef HAL_DCMI_DeInit (DCMI_HandleTypeDef * hdcmi)**
Function description	De-initialize the DCMI peripheral, reset control registers to their default values.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **HAL:** status

HAL_DCMI_MspInit

Function name	**void HAL_DCMI_MspInit (DCMI_HandleTypeDef * hdcmi)**
Function description	Initialize the DCMI MSP.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **None:**

HAL_DCMI_MspDeInit

Function name	**void HAL_DCMI_MspDeInit (DCMI_HandleTypeDef * hdcmi)**
Function description	De-initialize the DCMI MSP.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **None:**

HAL_DCMI_Start_DMA

Function name	**HAL_StatusTypeDef HAL_DCMI_Start_DMA (DCMI_HandleTypeDef * hdcmi, uint32_t DCMI_Mode, uint32_t pData, uint32_t Length)**
Function description	Enable DCMI capture in DMA mode.

Parameters	• **hdcmi:** Pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI. • **DCMI_Mode:** DCMI capture mode snapshot or continuous grab. • **pData:** The destination memory buffer address. • **Length:** The length of capture to be transferred (in 32-bit words).
Return values	• **HAL:** status
Notes	• In case of length larger than 65535 (0xFFFF is the DMA maximum transfer length), the API uses the end of the destination buffer as a work area: HAL_DCMI_Start_DMA() initiates a circular DMA transfer from DCMI DR to the ad-hoc work buffer and each half and complete transfer interrupt triggers a copy from the work buffer to the final destination pData thru a second DMA channel. • Following HAL_DCMI_Init() call, all interruptions are enabled (line end, frame end, overrun, VSYNC and embedded synchronization error interrupts). User can disable unwanted interrupts thru __HAL_DCMI_DISABLE_IT() macro before invoking HAL_DCMI_Start_DMA(). • For length less than 0xFFFF (DMA maximum transfer length) and in snapshot mode, frame interrupt is disabled before DMA transfer. FRAME capture flag is checked in DCMI_DMAXferCplt callback at the end of the DMA transfer. If flag is set, HAL_DCMI_FrameEventCallback() API is called.

HAL_DCMI_Stop

Function name	**HAL_StatusTypeDef HAL_DCMI_Stop (DCMI_HandleTypeDef * hdcmi)**
Function description	Disable DCMI capture in DMA mode.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **HAL:** status

HAL_DCMI_Suspend

Function name	**HAL_StatusTypeDef HAL_DCMI_Suspend (DCMI_HandleTypeDef * hdcmi)**
Function description	Suspend DCMI capture.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **HAL:** status

HAL_DCMI_Resume

Function name	**HAL_StatusTypeDef HAL_DCMI_Resume (DCMI_HandleTypeDef * hdcmi)**
Function description	Resume DCMI capture.

Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **HAL:** status

HAL_DCMI_ErrorCallback

Function name	**void HAL_DCMI_ErrorCallback (DCMI_HandleTypeDef * hdcmi)**
Function description	Error DCMI callback.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **None:**

HAL_DCMI_LineEventCallback

Function name	**void HAL_DCMI_LineEventCallback (DCMI_HandleTypeDef * hdcmi)**
Function description	Line Event callback.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **None:**

HAL_DCMI_FrameEventCallback

Function name	**void HAL_DCMI_FrameEventCallback (DCMI_HandleTypeDef * hdcmi)**
Function description	Frame Event callback.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **None:**

HAL_DCMI_VsyncEventCallback

Function name	**void HAL_DCMI_VsyncEventCallback (DCMI_HandleTypeDef * hdcmi)**
Function description	VSYNC Event callback.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **None:**

HAL_DCMI_IRQHandler

Function name	**void HAL_DCMI_IRQHandler (DCMI_HandleTypeDef * hdcmi)**
Function description	Handle DCMI interrupt request.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for the DCMI.

Return values	• **None:**

HAL_DCMI_ConfigCrop

Function name	**HAL_StatusTypeDef HAL_DCMI_ConfigCrop (DCMI_HandleTypeDef * hdcmi, uint32_t X0, uint32_t Y0, uint32_t XSize, uint32_t YSize)**
Function description	Configure the DCMI crop window coordinates.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI. • **X0:** DCMI window crop window X offset (number of pixels clocks to count before the capture). • **Y0:** DCMI window crop window Y offset (image capture starts with this line number, previous line data are ignored). • **XSize:** DCMI crop window horizontal size (in number of pixels per line). • **YSize:** DCMI crop window vertical size (in lines count).
Return values	• **HAL:** status
Notes	• For all the parameters, the actual value is the input data + 1 (e.g. YSize = 0x0 means 1 line, YSize = 0x1 means 2 lines, ...)

HAL_DCMI_EnableCrop

Function name	**HAL_StatusTypeDef HAL_DCMI_EnableCrop (DCMI_HandleTypeDef * hdcmi)**
Function description	Enable the crop feature.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **HAL:** status

HAL_DCMI_DisableCrop

Function name	**HAL_StatusTypeDef HAL_DCMI_DisableCrop (DCMI_HandleTypeDef * hdcmi)**
Function description	Disable the crop feature.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **HAL:** status

HAL_DCMI_ConfigSyncUnmask

Function name	**HAL_StatusTypeDef HAL_DCMI_ConfigSyncUnmask (DCMI_HandleTypeDef * hdcmi, DCMI_SyncUnmaskTypeDef * SyncUnmask)**
Function description	Set embedded synchronization delimiters unmasks.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

	•	**SyncUnmask:** pointer to a DCMI_SyncUnmaskTypeDef structure that contains the embedded synchronization delimiters unmasks.
Return values	•	**HAL:** status

HAL_DCMI_GetState

Function name	**HAL_DCMI_StateTypeDef HAL_DCMI_GetState (DCMI_HandleTypeDef * hdcmi)**
Function description	Return the DCMI state.
Parameters	• **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **HAL:** state

HAL_DCMI_GetError

Function name	**uint32_t HAL_DCMI_GetError (DCMI_HandleTypeDef * hdcmi)**
Function description	Return the DCMI error code.
Parameters	• **hdcmi::** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	• **DCMI:** Error Code

16.3 DCMI Firmware driver defines

16.3.1 DCMI

DCMI Byte Select Mode

DCMI_BSM_ALL	Interface captures all received data
DCMI_BSM_OTHER	Interface captures every other byte from the received data
DCMI_BSM_ALTERNATE_4	Interface captures one byte out of four
DCMI_BSM_ALTERNATE_2	Interface captures two bytes out of four

DCMI Byte Select Start

DCMI_OEBS_ODD	Interface captures first data from the frame/line start, second one being dropped
DCMI_OEBS_EVEN	Interface captures second data from the frame/line start, first one being dropped

DCMI Capture Mode

DCMI_MODE_CONTINUOUS	The received data are transferred continuously into the destination memory through the DMA
DCMI_MODE_SNAPSHOT	Once activated, the interface waits for the start of frame and then transfers a single frame through the DMA

DCMI Capture Rate

DCMI_CR_ALL_FRAME	All frames are captured

DCMI_CR_ALTERNATE_2_FRAME	Every alternate frame captured
DCMI_CR_ALTERNATE_4_FRAME	One frame in 4 frames captured

DCMI Error Code

HAL_DCMI_ERROR_NONE	No error
HAL_DCMI_ERROR_OVR	Overrun error
HAL_DCMI_ERROR_SYNC	Synchronization error
HAL_DCMI_ERROR_TIMEOUT	Timeout error
HAL_DCMI_ERROR_DMA	DMA error

DCMI Exported Macros

__HAL_DCMI_RESET_HANDLE_STATE

Description:

- Reset DCMI handle state.

Parameters:

- __HANDLE__: specifies the DCMI handle.

Return value:

- None

__HAL_DCMI_ENABLE

Description:

- Enable the DCMI.

Parameters:

- __HANDLE__: DCMI handle

Return value:

- None

__HAL_DCMI_DISABLE

Description:

- Disable the DCMI.

Parameters:

- __HANDLE__: DCMI handle

Return value:

- None

__HAL_DCMI_GET_FLAG

Description:

- Get the DCMI pending flag.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: Get the specified flag. This parameter can be one of the following values (no combination allowed)
  - DCMI_FLAG_HSYNC: HSYNC pin state (active line / synchronization between lines)
  - DCMI_FLAG_VSYNC: VSYNC pin

state (active frame / synchronization between frames)
  - DCMI_FLAG_FNE: FIFO empty flag
  - DCMI_FLAG_FRAMERI: Frame capture complete flag
  - DCMI_FLAG_OVRRI: Overrun flag
  - DCMI_FLAG_ERRRI: Synchronization error flag
  - DCMI_FLAG_VSYNCRI: VSYNC flag
  - DCMI_FLAG_LINERI: Line flag
  - DCMI_FLAG_FRAMEMI: DCMI Capture complete masked interrupt status
  - DCMI_FLAG_OVRMI: DCMI Overrun masked interrupt status
  - DCMI_FLAG_ERRMI: DCMI Synchronization error masked interrupt status
  - DCMI_FLAG_VSYNCMI: DCMI VSYNC masked interrupt status
  - DCMI_FLAG_LINEMI: DCMI Line masked interrupt status

Return value:

- The: state of FLAG.

__HAL_DCMI_CLEAR_FLAG

Description:

- Clear the DCMI pending flag.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DCMI_FLAG_FRAMERI: Frame capture complete flag
  - DCMI_FLAG_OVRRI: Overrun flag
  - DCMI_FLAG_ERRRI: Synchronization error flag
  - DCMI_FLAG_VSYNCRI: VSYNC flag
  - DCMI_FLAG_LINERI: Line flag

Return value:

- None

__HAL_DCMI_ENABLE_IT

Description:

- Enable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This

parameter can be any combination of the following values:
  – DCMI_IT_FRAME: Frame capture complete interrupt
  – DCMI_IT_OVR: Overrun interrupt
  – DCMI_IT_ERR: Synchronization error interrupt
  – DCMI_IT_VSYNC: VSYNC interrupt
  – DCMI_IT_LINE: Line interrupt

Return value:

- None

__HAL_DCMI_DISABLE_IT

Description:

- Disable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  – DCMI_IT_FRAME: Frame capture complete interrupt
  – DCMI_IT_OVR: Overrun interrupt
  – DCMI_IT_ERR: Synchronization error interrupt
  – DCMI_IT_VSYNC: VSYNC interrupt
  – DCMI_IT_LINE: Line interrupt

Return value:

- None

__HAL_DCMI_GET_IT_SOURCE

Description:

- Check whether or not the specified DCMI interrupt has occurred and that the interruption is enabled at the same time.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt flag and source to check. This parameter can be one of the following values:
  – DCMI_IT_FRAME: Frame capture complete interrupt mask
  – DCMI_IT_OVR: Overrun interrupt mask
  – DCMI_IT_ERR: Synchronization error interrupt mask
  – DCMI_IT_VSYNC: VSYNC interrupt mask
  – DCMI_IT_LINE: Line interrupt mask

Return value:

- The: state of INTERRUPT.

Notes:

- A bit in MIS register is set if the corresponding enable bit in DCMI_IER is set and the corresponding bit in DCMI_RIS is set.

DCMI Extended Data Mode

DCMI_EXTEND_DATA_8B	Interface captures 8-bit data on every pixel clock
DCMI_EXTEND_DATA_10B	Interface captures 10-bit data on every pixel clock
DCMI_EXTEND_DATA_12B	Interface captures 12-bit data on every pixel clock
DCMI_EXTEND_DATA_14B	Interface captures 14-bit data on every pixel clock

DCMI Flags

DCMI_FLAG_HSYNC	HSYNC pin state (active line / synchronization between lines)
DCMI_FLAG_VSYNC	VSYNC pin state (active frame / synchronization between frames)
DCMI_FLAG_FNE	FIFO not empty flag
DCMI_FLAG_FRAMERI	Capture complete interrupt flag
DCMI_FLAG_OVRRI	Overrun interrupt flag
DCMI_FLAG_ERRRI	Synchronization error interrupt flag
DCMI_FLAG_VSYNCRI	VSYNC interrupt flag
DCMI_FLAG_LINERI	Line interrupt flag
DCMI_FLAG_FRAMEMI	DCMI Capture complete masked interrupt status
DCMI_FLAG_OVRMI	DCMI Overrun masked interrupt status
DCMI_FLAG_ERRMI	DCMI Synchronization error masked interrupt status
DCMI_FLAG_VSYNCMI	DCMI VSYNC masked interrupt status
DCMI_FLAG_LINEMI	DCMI Line masked interrupt status

DCMI HSYNC Polarity

DCMI_HSPOLARITY_LOW	Horizontal synchronization active Low
DCMI_HSPOLARITY_HIGH	Horizontal synchronization active High

DCMI Interrupt Sources

DCMI_IT_FRAME	Capture complete interrupt
DCMI_IT_OVR	Overrun interrupt
DCMI_IT_ERR	Synchronization error interrupt
DCMI_IT_VSYNC	VSYNC interrupt
DCMI_IT_LINE	Line interrupt

DCMI JPEG Mode

DCMI_JPEG_DISABLE	JPEG mode disabled

DCMI_JPEG_ENABLE JPEG mode enabled

DCMI Line Select Mode

DCMI_LSM_ALL Interface captures all received lines

DCMI_LSM_ALTERNATE_2 Interface captures one line out of two

DCMI Line Select Start

DCMI_OELS_ODD Interface captures first line from the frame start, second one being dropped

DCMI_OELS_EVEN Interface captures second line from the frame start, first one being dropped

DCMI Pixel Clock Polarity

DCMI_PCKPOLARITY_FALLING Pixel clock active on Falling edge

DCMI_PCKPOLARITY_RISING Pixel clock active on Rising edge

DCMI Registers Indices

DCMI_MIS_INDEX DCMI MIS register index

DCMI_SR_INDEX DCMI SR register index

DCMI Shifts

DCMI_POSITION_CWSIZE_VLINE Required left shift to set crop window vertical line count

DCMI_POSITION_CWSTRT_VST Required left shift to set crop window vertical start line count

DCMI_POSITION_ESCR_LSC Required left shift to set line start delimiter

DCMI_POSITION_ESCR_LEC Required left shift to set line end delimiter

DCMI_POSITION_ESCR_FEC Required left shift to set frame end delimiter

DCMI_POSITION_ESUR_LSU Required left shift to set line start delimiter unmask

DCMI_POSITION_ESUR_LEU Required left shift to set line end delimiter unmask

DCMI_POSITION_ESUR_FEU Required left shift to set frame end delimiter unmask

DCMI Stop TimeOut

DCMI_TIMEOUT_STOP 1s

DCMI Synchronization Mode

DCMI_SYNCHRO_HARDWARE Hardware synchronization data capture (frame/line start/stop) is synchronized with the HSYNC/VSYNC signals

DCMI_SYNCHRO_EMBEDDED Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow

DCMI VSYNC Polarity

DCMI_VSPOLARITY_LOW Vertical synchronization active Low

DCMI_VSPOLARITY_HIGH Vertical synchronization active High

DCMI Window Coordinate

DCMI_WINDOW_COORDINATE Window coordinate

DCMI Window Height

DCMI_WINDOW_HEIGHT Window Height

17 HAL DFSDM Generic Driver

17.1 DFSDM Firmware driver registers structures

17.1.1 DFSDM_Channel_OutputClockTypeDef

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Selection***
- ***uint32_t Divider***

Field Documentation

- ***FunctionalState DFSDM_Channel_OutputClockTypeDef::Activation***
  Output clock enable/disable
- ***uint32_t DFSDM_Channel_OutputClockTypeDef::Selection***
  Output clock is system clock or audio clock. This parameter can be a value of ***DFSDM_Channel_OuputClock***
- ***uint32_t DFSDM_Channel_OutputClockTypeDef::Divider***
  Output clock divider. This parameter must be a number between Min_Data = 2 and Max_Data = 256

17.1.2 DFSDM_Channel_InputTypeDef

Data Fields

- ***uint32_t Multiplexer***
- ***uint32_t DataPacking***
- ***uint32_t Pins***

Field Documentation

- ***uint32_t DFSDM_Channel_InputTypeDef::Multiplexer***
  Input is external serial inputs, internal register or ADC output. ADC output is available only on STM32L451xx, STM32L452xx, STM32L462xx, STM32L496xx, STM32L4A6xx, STM32L4R5xx, STM32L4R7xx, STM32L4R9xx, STM32L4S5xx, STM32L4S7xx and STM32L4S9xx products. This parameter can be a value of ***DFSDM_Channel_InputMultiplexer***
- ***uint32_t DFSDM_Channel_InputTypeDef::DataPacking***
  Standard, interleaved or dual mode for internal register. This parameter can be a value of ***DFSDM_Channel_DataPacking***
- ***uint32_t DFSDM_Channel_InputTypeDef::Pins***
  Input pins are taken from same or following channel. This parameter can be a value of ***DFSDM_Channel_InputPins***

17.1.3 DFSDM_Channel_SerialInterfaceTypeDef

Data Fields

- ***uint32_t Type***
- ***uint32_t SpiClock***

Field Documentation

- ***uint32_t DFSDM_Channel_SerialInterfaceTypeDef::Type***
  SPI or Manchester modes. This parameter can be a value of
  DFSDM_Channel_SerialInterfaceType
- ***uint32_t DFSDM_Channel_SerialInterfaceTypeDef::SpiClock***
  SPI clock select (external or internal with different sampling point). This parameter can be a value of ***DFSDM_Channel_SpiClock***

17.1.4 DFSDM_Channel_AwdTypeDef

Data Fields

- ***uint32_t FilterOrder***
- ***uint32_t Oversampling***

Field Documentation

- ***uint32_t DFSDM_Channel_AwdTypeDef::FilterOrder***
  Analog watchdog Sinc filter order. This parameter can be a value of
  DFSDM_Channel_AwdFilterOrder
- ***uint32_t DFSDM_Channel_AwdTypeDef::Oversampling***
  Analog watchdog filter oversampling ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 32

17.1.5 DFSDM_Channel_InitTypeDef

Data Fields

- ***DFSDM_Channel_OutputClockTypeDef OutputClock***
- ***DFSDM_Channel_InputTypeDef Input***
- ***DFSDM_Channel_SerialInterfaceTypeDef SerialInterface***
- ***DFSDM_Channel_AwdTypeDef Awd***
- ***int32_t Offset***
- ***uint32_t RightBitShift***

Field Documentation

- ***DFSDM_Channel_OutputClockTypeDef DFSDM_Channel_InitTypeDef::OutputClock***
  DFSDM channel output clock parameters
- ***DFSDM_Channel_InputTypeDef DFSDM_Channel_InitTypeDef::Input***
  DFSDM channel input parameters
- ***DFSDM_Channel_SerialInterfaceTypeDef DFSDM_Channel_InitTypeDef::SerialInterface***
  DFSDM channel serial interface parameters
- ***DFSDM_Channel_AwdTypeDef DFSDM_Channel_InitTypeDef::Awd***
  DFSDM channel analog watchdog parameters
- ***int32_t DFSDM_Channel_InitTypeDef::Offset***
  DFSDM channel offset. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607
- ***uint32_t DFSDM_Channel_InitTypeDef::RightBitShift***
  DFSDM channel right bit shift. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F

17.1.6 DFSDM_Channel_HandleTypeDef

Data Fields

- ***DFSDM_Channel_TypeDef * Instance***
- ***DFSDM_Channel_InitTypeDef Init***

- ***HAL_DFSDM_Channel_StateTypeDef State***

Field Documentation

- ***DFSDM_Channel_TypeDef* DFSDM_Channel_HandleTypeDef::Instance***
  DFSDM channel instance
- ***DFSDM_Channel_InitTypeDef DFSDM_Channel_HandleTypeDef::Init***
  DFSDM channel init parameters
- ***HAL_DFSDM_Channel_StateTypeDef DFSDM_Channel_HandleTypeDef::State***
  DFSDM channel state

17.1.7 DFSDM_Filter_RegularParamTypeDef

Data Fields

- ***uint32_t Trigger***
- ***FunctionalState FastMode***
- ***FunctionalState DmaMode***

Field Documentation

- ***uint32_t DFSDM_Filter_RegularParamTypeDef::Trigger***
  Trigger used to start regular conversion: software or synchronous. This parameter can be a value of ***DFSDM_Filter_Trigger***
- ***FunctionalState DFSDM_Filter_RegularParamTypeDef::FastMode***
  Enable/disable fast mode for regular conversion
- ***FunctionalState DFSDM_Filter_RegularParamTypeDef::DmaMode***
  Enable/disable DMA for regular conversion

17.1.8 DFSDM_Filter_InjectedParamTypeDef

Data Fields

- ***uint32_t Trigger***
- ***FunctionalState ScanMode***
- ***FunctionalState DmaMode***
- ***uint32_t ExtTrigger***
- ***uint32_t ExtTriggerEdge***

Field Documentation

- ***uint32_t DFSDM_Filter_InjectedParamTypeDef::Trigger***
  Trigger used to start injected conversion: software, external or synchronous. This parameter can be a value of ***DFSDM_Filter_Trigger***
- ***FunctionalState DFSDM_Filter_InjectedParamTypeDef::ScanMode***
  Enable/disable scanning mode for injected conversion
- ***FunctionalState DFSDM_Filter_InjectedParamTypeDef::DmaMode***
  Enable/disable DMA for injected conversion
- ***uint32_t DFSDM_Filter_InjectedParamTypeDef::ExtTrigger***
  External trigger. This parameter can be a value of ***DFSDM_Filter_ExtTrigger***
- ***uint32_t DFSDM_Filter_InjectedParamTypeDef::ExtTriggerEdge***
  External trigger edge: rising, falling or both. This parameter can be a value of ***DFSDM_Filter_ExtTriggerEdge***

17.1.9 DFSDM_Filter_FilterParamTypeDef

Data Fields

- ***uint32_t SincOrder***
- ***uint32_t Oversampling***

- ***uint32_t IntOversampling***

Field Documentation

- ***uint32_t DFSDM_Filter_FilterParamTypeDef::SincOrder***
  Sinc filter order. This parameter can be a value of ***DFSDM_Filter_SincOrder***
- ***uint32_t DFSDM_Filter_FilterParamTypeDef::Oversampling***
  Filter oversampling ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 1024
- ***uint32_t DFSDM_Filter_FilterParamTypeDef::IntOversampling***
  Integrator oversampling ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 256

17.1.10 DFSDM_Filter_InitTypeDef

Data Fields

- ***DFSDM_Filter_RegularParamTypeDef RegularParam***
- ***DFSDM_Filter_InjectedParamTypeDef InjectedParam***
- ***DFSDM_Filter_FilterParamTypeDef FilterParam***

Field Documentation

- ***DFSDM_Filter_RegularParamTypeDef DFSDM_Filter_InitTypeDef::RegularParam***
  DFSDM regular conversion parameters
- ***DFSDM_Filter_InjectedParamTypeDef DFSDM_Filter_InitTypeDef::InjectedParam***
  DFSDM injected conversion parameters
- ***DFSDM_Filter_FilterParamTypeDef DFSDM_Filter_InitTypeDef::FilterParam***
  DFSDM filter parameters

17.1.11 DFSDM_Filter_HandleTypeDef

Data Fields

- ***DFSDM_Filter_TypeDef * Instance***
- ***DFSDM_Filter_InitTypeDef Init***
- ***DMA_HandleTypeDef * hdmaReg***
- ***DMA_HandleTypeDef * hdmaInj***
- ***uint32_t RegularContMode***
- ***uint32_t RegularTrigger***
- ***uint32_t InjectedTrigger***
- ***uint32_t ExtTriggerEdge***
- ***FunctionalState InjectedScanMode***
- ***uint32_t InjectedChannelsNbr***
- ***uint32_t InjConvRemaining***
- ***HAL_DFSDM_Filter_StateTypeDef State***
- ***uint32_t ErrorCode***

Field Documentation

- ***DFSDM_Filter_TypeDef* DFSDM_Filter_HandleTypeDef::Instance***
  DFSDM filter instance
- ***DFSDM_Filter_InitTypeDef DFSDM_Filter_HandleTypeDef::Init***
  DFSDM filter init parameters
- ***DMA_HandleTypeDef* DFSDM_Filter_HandleTypeDef::hdmaReg***
  Pointer on DMA handler for regular conversions
- ***DMA_HandleTypeDef* DFSDM_Filter_HandleTypeDef::hdmaInj***
  Pointer on DMA handler for injected conversions

- ***uint32_t DFSDM_Filter_HandleTypeDef::RegularContMode***
  Regular conversion continuous mode
- ***uint32_t DFSDM_Filter_HandleTypeDef::RegularTrigger***
  Trigger used for regular conversion
- ***uint32_t DFSDM_Filter_HandleTypeDef::InjectedTrigger***
  Trigger used for injected conversion
- ***uint32_t DFSDM_Filter_HandleTypeDef::ExtTriggerEdge***
  Rising, falling or both edges selected
- ***FunctionalState DFSDM_Filter_HandleTypeDef::InjectedScanMode***
  Injected scanning mode
- ***uint32_t DFSDM_Filter_HandleTypeDef::InjectedChannelsNbr***
  Number of channels in injected sequence
- ***uint32_t DFSDM_Filter_HandleTypeDef::InjConvRemaining***
  Injected conversions remaining
- ***HAL_DFSDM_Filter_StateTypeDef DFSDM_Filter_HandleTypeDef::State***
  DFSDM filter state
- ***uint32_t DFSDM_Filter_HandleTypeDef::ErrorCode***
  DFSDM filter error code

17.1.12 DFSDM_Filter_AwdParamTypeDef

Data Fields

- ***uint32_t DataSource***
- ***uint32_t Channel***
- ***int32_t HighThreshold***
- ***int32_t LowThreshold***
- ***uint32_t HighBreakSignal***
- ***uint32_t LowBreakSignal***

Field Documentation

- ***uint32_t DFSDM_Filter_AwdParamTypeDef::DataSource***
  Values from digital filter or from channel watchdog filter. This parameter can be a value of ***DFSDM_Filter_AwdDataSource***
- ***uint32_t DFSDM_Filter_AwdParamTypeDef::Channel***
  Analog watchdog channel selection. This parameter can be a values combination of ***DFSDM_Channel_Selection***
- ***int32_t DFSDM_Filter_AwdParamTypeDef::HighThreshold***
  High threshold for the analog watchdog. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607
- ***int32_t DFSDM_Filter_AwdParamTypeDef::LowThreshold***
  Low threshold for the analog watchdog. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607
- ***uint32_t DFSDM_Filter_AwdParamTypeDef::HighBreakSignal***
  Break signal assigned to analog watchdog high threshold event. This parameter can be a values combination of ***DFSDM_BreakSignals***
- ***uint32_t DFSDM_Filter_AwdParamTypeDef::LowBreakSignal***
  Break signal assigned to analog watchdog low threshold event. This parameter can be a values combination of ***DFSDM_BreakSignals***

17.2 DFSDM Firmware driver API description

17.2.1 How to use this driver

Channel initialization

1. User has first to initialize channels (before filters initialization).
2. As prerequisite, fill in the HAL_DFSDM_ChannelMspInit():
   - Enable DFSDMz clock interface with __HAL_RCC_DFSDMz_CLK_ENABLE().
   - Enable the clocks for the DFSDMz GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
   - Configure these DFSDMz pins in alternate mode using HAL_GPIO_Init().
   - If interrupt mode is used, enable and configure DFSDMz_FLT0 global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
3. Configure the output clock, input, serial interface, analog watchdog, offset and data right bit shift parameters for this channel using the HAL_DFSDM_ChannelInit() function.

Channel clock absence detector

1. Start clock absence detector using HAL_DFSDM_ChannelCkabStart() or HAL_DFSDM_ChannelCkabStart_IT().
2. In polling mode, use HAL_DFSDM_ChannelPollForCkab() to detect the clock absence.
3. In interrupt mode, HAL_DFSDM_ChannelCkabCallback() will be called if clock absence is detected.
4. Stop clock absence detector using HAL_DFSDM_ChannelCkabStop() or HAL_DFSDM_ChannelCkabStop_IT().
5. Please note that the same mode (polling or interrupt) has to be used for all channels because the channels are sharing the same interrupt.
6. Please note also that in interrupt mode, if clock absence detector is stopped for one channel, interrupt will be disabled for all channels.

Channel short circuit detector

1. Start short circuit detector using HAL_DFSDM_ChannelScdStart() or or HAL_DFSDM_ChannelScdStart_IT().
2. In polling mode, use HAL_DFSDM_ChannelPollForScd() to detect short circuit.
3. In interrupt mode, HAL_DFSDM_ChannelScdCallback() will be called if short circuit is detected.
4. Stop short circuit detector using HAL_DFSDM_ChannelScdStop() or or HAL_DFSDM_ChannelScdStop_IT().
5. Please note that the same mode (polling or interrupt) has to be used for all channels because the channels are sharing the same interrupt.
6. Please note also that in interrupt mode, if short circuit detector is stopped for one channel, interrupt will be disabled for all channels.

Channel analog watchdog value

1. Get analog watchdog filter value of a channel using HAL_DFSDM_ChannelGetAwdValue().

Channel offset value

1. Modify offset value of a channel using HAL_DFSDM_ChannelModifyOffset().

Filter initialization

1. After channel initialization, user has to init filters.
2. As prerequisite, fill in the HAL_DFSDM_FilterMspInit():
   – If interrupt mode is used , enable and configure DFSDMz_FLTx global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ(). Please note that DFSDMz_FLT0 global interrupt could be already enabled if interrupt is used for channel.
   – If DMA mode is used, configure DMA with HAL_DMA_Init() and link it with DFSDMz filter handle using __HAL_LINKDMA().
3. Configure the regular conversion, injected conversion and filter parameters for this filter using the HAL_DFSDM_FilterInit() function.

Filter regular channel conversion

1. Select regular channel and enable/disable continuous mode using HAL_DFSDM_FilterConfigRegChannel().
2. Start regular conversion using HAL_DFSDM_FilterRegularStart(), HAL_DFSDM_FilterRegularStart_IT(), HAL_DFSDM_FilterRegularStart_DMA() or HAL_DFSDM_FilterRegularMsbStart_DMA().
3. In polling mode, use HAL_DFSDM_FilterPollForRegConversion() to detect the end of regular conversion.
4. In interrupt mode, HAL_DFSDM_FilterRegConvCpltCallback() will be called at the end of regular conversion.
5. Get value of regular conversion and corresponding channel using HAL_DFSDM_FilterGetRegularValue().
6. In DMA mode, HAL_DFSDM_FilterRegConvHalfCpltCallback() and HAL_DFSDM_FilterRegConvCpltCallback() will be called respectively at the half transfer and at the transfer complete. Please note that HAL_DFSDM_FilterRegConvHalfCpltCallback() will be called only in DMA circular mode.
7. Stop regular conversion using HAL_DFSDM_FilterRegularStop(), HAL_DFSDM_FilterRegularStop_IT() or HAL_DFSDM_FilterRegularStop_DMA().

Filter injected channels conversion

1. Select injected channels using HAL_DFSDM_FilterConfigInjChannel().
2. Start injected conversion using HAL_DFSDM_FilterInjectedStart(), HAL_DFSDM_FilterInjectedStart_IT(), HAL_DFSDM_FilterInjectedStart_DMA() or HAL_DFSDM_FilterInjectedMsbStart_DMA().
3. In polling mode, use HAL_DFSDM_FilterPollForInjConversion() to detect the end of injected conversion.
4. In interrupt mode, HAL_DFSDM_FilterInjConvCpltCallback() will be called at the end of injected conversion.
5. Get value of injected conversion and corresponding channel using HAL_DFSDM_FilterGetInjectedValue().
6. In DMA mode, HAL_DFSDM_FilterInjConvHalfCpltCallback() and HAL_DFSDM_FilterInjConvCpltCallback() will be called respectively at the half transfer and at the transfer complete. Please note that HAL_DFSDM_FilterInjConvCpltCallback() will be called only in DMA circular mode.
7. Stop injected conversion using HAL_DFSDM_FilterInjectedStop(), HAL_DFSDM_FilterInjectedStop_IT() or HAL_DFSDM_FilterInjectedStop_DMA().

Filter analog watchdog

1. Start filter analog watchdog using HAL_DFSDM_FilterAwdStart_IT().
2. HAL_DFSDM_FilterAwdCallback() will be called if analog watchdog occurs.

3. Stop filter analog watchdog using HAL_DFSDM_FilterAwdStop_IT().

Filter extreme detector

1. Start filter extreme detector using HAL_DFSDM_FilterExdStart().
2. Get extreme detector maximum value using HAL_DFSDM_FilterGetExdMaxValue().
3. Get extreme detector minimum value using HAL_DFSDM_FilterGetExdMinValue().
4. Start filter extreme detector using HAL_DFSDM_FilterExdStop().

Filter conversion time

1. Get conversion time value using HAL_DFSDM_FilterGetConvTimeValue().

17.2.2 Channel initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the DFSDM channel.
- De-initialize the DFSDM channel.

This section contains the following APIs:

- ***HAL_DFSDM_ChannelInit()***
- ***HAL_DFSDM_ChannelDeInit()***
- ***HAL_DFSDM_ChannelMspInit()***
- ***HAL_DFSDM_ChannelMspDeInit()***

17.2.3 Channel operation functions

This section provides functions allowing to:

- Manage clock absence detector feature.
- Manage short circuit detector feature.
- Get analog watchdog value.
- Modify offset value.

This section contains the following APIs:

- ***HAL_DFSDM_ChannelCkabStart()***
- ***HAL_DFSDM_ChannelPollForCkab()***
- ***HAL_DFSDM_ChannelCkabStop()***
- ***HAL_DFSDM_ChannelCkabStart_IT()***
- ***HAL_DFSDM_ChannelCkabCallback()***
- ***HAL_DFSDM_ChannelCkabStop_IT()***
- ***HAL_DFSDM_ChannelScdStart()***
- ***HAL_DFSDM_ChannelPollForScd()***
- ***HAL_DFSDM_ChannelScdStop()***
- ***HAL_DFSDM_ChannelScdStart_IT()***
- ***HAL_DFSDM_ChannelScdCallback()***
- ***HAL_DFSDM_ChannelScdStop_IT()***
- ***HAL_DFSDM_ChannelGetAwdValue()***
- ***HAL_DFSDM_ChannelModifyOffset()***

17.2.4 Channel state function

This section provides function allowing to:

- Get channel handle state.

This section contains the following APIs:

- *HAL_DFSDM_ChannelGetState()*

17.2.5 Filter initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the DFSDM filter.
- De-initialize the DFSDM filter.

This section contains the following APIs:

- *HAL_DFSDM_FilterInit()*
- *HAL_DFSDM_FilterDeInit()*
- *HAL_DFSDM_FilterMspInit()*
- *HAL_DFSDM_FilterMspDeInit()*

17.2.6 Filter control functions

This section provides functions allowing to:

- Select channel and enable/disable continuous mode for regular conversion.
- Select channels for injected conversion.

This section contains the following APIs:

- *HAL_DFSDM_FilterConfigRegChannel()*
- *HAL_DFSDM_FilterConfigInjChannel()*

17.2.7 Filter operation functions

This section provides functions allowing to:

- Start conversion of regular/injected channel.
- Poll for the end of regular/injected conversion.
- Stop conversion of regular/injected channel.
- Start conversion of regular/injected channel and enable interrupt.
- Call the callback functions at the end of regular/injected conversions.
- Stop conversion of regular/injected channel and disable interrupt.
- Start conversion of regular/injected channel and enable DMA transfer.
- Stop conversion of regular/injected channel and disable DMA transfer.
- Start analog watchdog and enable interrupt.
- Call the callback function when analog watchdog occurs.
- Stop analog watchdog and disable interrupt.
- Start extreme detector.
- Stop extreme detector.
- Get result of regular channel conversion.
- Get result of injected channel conversion.
- Get extreme detector maximum and minimum values.
- Get conversion time.
- Handle DFSDM interrupt request.

This section contains the following APIs:

- *HAL_DFSDM_FilterRegularStart()*
- *HAL_DFSDM_FilterPollForRegConversion()*
- *HAL_DFSDM_FilterRegularStop()*
- *HAL_DFSDM_FilterRegularStart_IT()*
- *HAL_DFSDM_FilterRegularStop_IT()*
- *HAL_DFSDM_FilterRegularStart_DMA()*

- *HAL_DFSDM_FilterRegularMsbStart_DMA()*
- *HAL_DFSDM_FilterRegularStop_DMA()*
- *HAL_DFSDM_FilterGetRegularValue()*
- *HAL_DFSDM_FilterInjectedStart()*
- *HAL_DFSDM_FilterPollForInjConversion()*
- *HAL_DFSDM_FilterInjectedStop()*
- *HAL_DFSDM_FilterInjectedStart_IT()*
- *HAL_DFSDM_FilterInjectedStop_IT()*
- *HAL_DFSDM_FilterInjectedStart_DMA()*
- *HAL_DFSDM_FilterInjectedMsbStart_DMA()*
- *HAL_DFSDM_FilterInjectedStop_DMA()*
- *HAL_DFSDM_FilterGetInjectedValue()*
- *HAL_DFSDM_FilterAwdStart_IT()*
- *HAL_DFSDM_FilterAwdStop_IT()*
- *HAL_DFSDM_FilterExdStart()*
- *HAL_DFSDM_FilterExdStop()*
- *HAL_DFSDM_FilterGetExdMaxValue()*
- *HAL_DFSDM_FilterGetExdMinValue()*
- *HAL_DFSDM_FilterGetConvTimeValue()*
- *HAL_DFSDM_IRQHandler()*
- *HAL_DFSDM_FilterRegConvCpltCallback()*
- *HAL_DFSDM_FilterRegConvHalfCpltCallback()*
- *HAL_DFSDM_FilterInjConvCpltCallback()*
- *HAL_DFSDM_FilterInjConvHalfCpltCallback()*
- *HAL_DFSDM_FilterAwdCallback()*
- *HAL_DFSDM_FilterErrorCallback()*

17.2.8 Filter state functions

This section provides functions allowing to:

- Get the DFSDM filter state.
- Get the DFSDM filter error.

This section contains the following APIs:

- *HAL_DFSDM_FilterGetState()*
- *HAL_DFSDM_FilterGetError()*

17.2.9 Detailed description of functions

HAL_DFSDM_ChannelInit

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelInit (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	Initialize the DFSDM channel according to the specified parameters in the DFSDM_ChannelInitTypeDef structure and initialize the associated handle.
Parameters	• **hdfsdm_channel::** DFSDM channel handle.
Return values	• **HAL:** status.

HAL_DFSDM_ChannelDeInit

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelDeInit (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	De-initialize the DFSDM channel.
Parameters	• **hdfsdm_channel::** DFSDM channel handle.
Return values	• **HAL:** status.

HAL_DFSDM_ChannelMspInit

Function name	**void HAL_DFSDM_ChannelMspInit (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	Initialize the DFSDM channel MSP.
Parameters	• **hdfsdm_channel::** DFSDM channel handle.
Return values	• **None:**

HAL_DFSDM_ChannelMspDeInit

Function name	**void HAL_DFSDM_ChannelMspDeInit (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	De-initialize the DFSDM channel MSP.
Parameters	• **hdfsdm_channel::** DFSDM channel handle.
Return values	• **None:**

HAL_DFSDM_ChannelCkabStart

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStart (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	This function allows to start clock absence detection in polling mode.
Parameters	• **hdfsdm_channel::** DFSDM channel handle.
Return values	• **HAL:** status
Notes	• Same mode has to be used for all channels. • If clock is not available on this channel during 5 seconds, clock absence detection will not be activated and function will return HAL_TIMEOUT error.

HAL_DFSDM_ChannelCkabStart_IT

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStart_IT (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	This function allows to start clock absence detection in interrupt mode.
Parameters	• **hdfsdm_channel::** DFSDM channel handle.
Return values	• **HAL:** status

Notes	• Same mode has to be used for all channels. • If clock is not available on this channel during 5 seconds, clock absence detection will not be activated and function will return HAL_TIMEOUT error.

HAL_DFSDM_ChannelCkabStop

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStop (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	This function allows to stop clock absence detection in polling mode.
Parameters	• **hdfsdm_channel::** DFSDM channel handle.
Return values	• **HAL:** status

HAL_DFSDM_ChannelCkabStop_IT

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStop_IT (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	This function allows to stop clock absence detection in interrupt mode.
Parameters	• **hdfsdm_channel::** DFSDM channel handle.
Return values	• **HAL:** status
Notes	• Interrupt will be disabled for all channels

HAL_DFSDM_ChannelScdStart

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelScdStart (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t Threshold, uint32_t BreakSignal)**
Function description	This function allows to start short circuit detection in polling mode.
Parameters	• **hdfsdm_channel::** DFSDM channel handle. • **Threshold:** Short circuit detector threshold. This parameter must be a number between Min_Data = 0 and Max_Data = 255. • **BreakSignal:** : Break signals assigned to short circuit event. This parameter can be a values combination of DFSDM break signals.
Return values	• **HAL:** status
Notes	• Same mode has to be used for all channels

HAL_DFSDM_ChannelScdStart_IT

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelScdStart_IT (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t Threshold, uint32_t BreakSignal)**
Function description	This function allows to start short circuit detection in interrupt mode.

Parameters	• **hdfsdm_channel:** : DFSDM channel handle. • **Threshold:** : Short circuit detector threshold. This parameter must be a number between Min_Data = 0 and Max_Data = 255. • **BreakSignal:** : Break signals assigned to short circuit event. This parameter can be a values combination of DFSDM break signals.
Return values	• **HAL:** status
Notes	• Same mode has to be used for all channels

HAL_DFSDM_ChannelScdStop

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelScdStop (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	This function allows to stop short circuit detection in polling mode.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle.
Return values	• **HAL:** status

HAL_DFSDM_ChannelScdStop_IT

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelScdStop_IT (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	This function allows to stop short circuit detection in interrupt mode.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle.
Return values	• **HAL:** status
Notes	• Interrupt will be disabled for all channels

HAL_DFSDM_ChannelGetAwdValue

Function name	**int16_t HAL_DFSDM_ChannelGetAwdValue (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	This function allows to get channel analog watchdog value.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle.
Return values	• **Channel:** analog watchdog value.

HAL_DFSDM_ChannelModifyOffset

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelModifyOffset (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, int32_t Offset)**
Function description	This function allows to modify channel offset value.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle. • **Offset:** : DFSDM channel offset. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607.

Return values	• **HAL:** status.

HAL_DFSDM_ChannelPollForCkab

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelPollForCkab (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t Timeout)**
Function description	This function allows to poll for the clock absence detection.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle. • **Timeout:** : Timeout value in milliseconds.
Return values	• **HAL:** status

HAL_DFSDM_ChannelPollForScd

Function name	**HAL_StatusTypeDef HAL_DFSDM_ChannelPollForScd (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t Timeout)**
Function description	This function allows to poll for the short circuit detection.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle. • **Timeout:** : Timeout value in milliseconds.
Return values	• **HAL:** status

HAL_DFSDM_ChannelCkabCallback

Function name	**void HAL_DFSDM_ChannelCkabCallback (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	Clock absence detection callback.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle.
Return values	• **None:**

HAL_DFSDM_ChannelScdCallback

Function name	**void HAL_DFSDM_ChannelScdCallback (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	Short circuit detection callback.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle.
Return values	• **None:**

HAL_DFSDM_ChannelGetState

Function name	**HAL_DFSDM_Channel_StateTypeDef HAL_DFSDM_ChannelGetState (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)**
Function description	This function allows to get the current DFSDM channel handle state.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle.

Return values	• **DFSDM:** channel state.

HAL_DFSDM_FilterInit

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterInit (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	Initialize the DFSDM filter according to the specified parameters in the DFSDM_FilterInitTypeDef structure and initialize the associated handle.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status.

HAL_DFSDM_FilterDeInit

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterDeInit (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	De-initializes the DFSDM filter.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status.

HAL_DFSDM_FilterMspInit

Function name	**void HAL_DFSDM_FilterMspInit (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	Initializes the DFSDM filter MSP.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **None:**

HAL_DFSDM_FilterMspDeInit

Function name	**void HAL_DFSDM_FilterMspDeInit (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	De-initializes the DFSDM filter MSP.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **None:**

HAL_DFSDM_FilterConfigRegChannel

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterConfigRegChannel (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Channel, uint32_t ContinuousMode)**
Function description	This function allows to select channel and to enable/disable continuous mode for regular conversion.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **Channel:** : Channel for regular conversion. This parameter can be a value of DFSDM Channel Selection. • **ContinuousMode:** : Enable/disable continuous mode for

regular conversion. This parameter can be a value of DFSDM Continuous Mode.

Return values	• **HAL:** status

HAL_DFSDM_FilterConfigInjChannel

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterConfigInjChannel (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Channel)**
Function description	This function allows to select channels for injected conversion.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **Channel:** : Channels for injected conversion. This parameter can be a values combination of DFSDM Channel Selection.
Return values	• **HAL:** status

HAL_DFSDM_FilterRegularStart

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterRegularStart (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to start regular conversion in polling mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status
Notes	• This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing.

HAL_DFSDM_FilterRegularStart_IT

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterRegularStart_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to start regular conversion in interrupt mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status
Notes	• This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing.

HAL_DFSDM_FilterRegularStart_DMA

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterRegularStart_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, int32_t * pData, uint32_t Length)**
Function description	This function allows to start regular conversion in DMA mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **pData:** : The destination buffer address. • **Length:** : The length of data to be transferred from DFSDM filter to memory.
Return values	• **HAL:** status

Notes	• This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing. Please note that data on buffer will contain signed regular conversion value on 24 most significant bits and corresponding channel on 3 least significant bits.

HAL_DFSDM_FilterRegularMsbStart_DMA

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterRegularMsbStart_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, int16_t * pData, uint32_t Length)**
Function description	This function allows to start regular conversion in DMA mode and to get only the 16 most significant bits of conversion.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **pData:** : The destination buffer address. • **Length:** : The length of data to be transferred from DFSDM filter to memory.
Return values	• **HAL:** status
Notes	• This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing. Please note that data on buffer will contain signed 16 most significant bits of regular conversion.

HAL_DFSDM_FilterRegularStop

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterRegularStop (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to stop regular conversion in polling mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status
Notes	• This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterRegularStop_IT

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterRegularStop_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to stop regular conversion in interrupt mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status
Notes	• This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterRegularStop_DMA

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterRegularStop_DMA**

	(DFSDM_Filter_HandleTypeDef * hdfsdm_filter)
Function description	This function allows to stop regular conversion in DMA mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status
Notes	• This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterInjectedStart

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStart (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to start injected conversion in polling mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status
Notes	• This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing.

HAL_DFSDM_FilterInjectedStart_IT

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStart_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to start injected conversion in interrupt mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status
Notes	• This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing.

HAL_DFSDM_FilterInjectedStart_DMA

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStart_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, int32_t * pData, uint32_t Length)**
Function description	This function allows to start injected conversion in DMA mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **pData:** : The destination buffer address. • **Length:** : The length of data to be transferred from DFSDM filter to memory.
Return values	• **HAL:** status
Notes	• This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing. Please note that data on buffer will contain signed injected conversion value on 24 most significant bits and corresponding channel on 3 least significant bits.

HAL_DFSDM_FilterInjectedMsbStart_DMA

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterInjectedMsbStart_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, int16_t * pData, uint32_t Length)**
Function description	This function allows to start injected conversion in DMA mode and to get only the 16 most significant bits of conversion.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **pData:** : The destination buffer address. • **Length:** : The length of data to be transferred from DFSDM filter to memory.
Return values	• **HAL:** status
Notes	• This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing. Please note that data on buffer will contain signed 16 most significant bits of injected conversion.

HAL_DFSDM_FilterInjectedStop

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStop (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to stop injected conversion in polling mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status
Notes	• This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterInjectedStop_IT

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStop_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to stop injected conversion in interrupt mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status
Notes	• This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterInjectedStop_DMA

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStop_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to stop injected conversion in DMA mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status
Notes	• This function should be called only if injected conversion is

ongoing.

HAL_DFSDM_FilterAwdStart_IT

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterAwdStart_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, DFSDM_Filter_AwdParamTypeDef * awdParam)**
Function description	This function allows to start filter analog watchdog in interrupt mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **awdParam:** : DFSDM filter analog watchdog parameters.
Return values	• **HAL:** status

HAL_DFSDM_FilterAwdStop_IT

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterAwdStop_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to stop filter analog watchdog in interrupt mode.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status

HAL_DFSDM_FilterExdStart

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterExdStart (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Channel)**
Function description	This function allows to start extreme detector feature.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **Channel:** : Channels where extreme detector is enabled. This parameter can be a values combination of DFSDM Channel Selection.
Return values	• **HAL:** status

HAL_DFSDM_FilterExdStop

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterExdStop (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to stop extreme detector feature.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **HAL:** status

HAL_DFSDM_FilterGetRegularValue

Function name	**int32_t HAL_DFSDM_FilterGetRegularValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)**

Function description	This function allows to get regular conversion value.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **Channel:** : Corresponding channel of regular conversion.
Return values	• **Regular:** conversion value

HAL_DFSDM_FilterGetInjectedValue

Function name	**int32_t HAL_DFSDM_FilterGetInjectedValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)**
Function description	This function allows to get injected conversion value.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **Channel:** : Corresponding channel of injected conversion.
Return values	• **Injected:** conversion value

HAL_DFSDM_FilterGetExdMaxValue

Function name	**int32_t HAL_DFSDM_FilterGetExdMaxValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)**
Function description	This function allows to get extreme detector maximum value.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **Channel:** : Corresponding channel.
Return values	• **Extreme:** detector maximum value This value is between Min_Data = -8388608 and Max_Data = 8388607.

HAL_DFSDM_FilterGetExdMinValue

Function name	**int32_t HAL_DFSDM_FilterGetExdMinValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)**
Function description	This function allows to get extreme detector minimum value.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **Channel:** : Corresponding channel.
Return values	• **Extreme:** detector minimum value This value is between Min_Data = -8388608 and Max_Data = 8388607.

HAL_DFSDM_FilterGetConvTimeValue

Function name	**uint32_t HAL_DFSDM_FilterGetConvTimeValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to get conversion time value.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **Conversion:** time value
Notes	• To get time in second, this value has to be divided by DFSDM clock frequency.

HAL_DFSDM_IRQHandler

Function name	**void HAL_DFSDM_IRQHandler (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function handles the DFSDM interrupts.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **None:**

HAL_DFSDM_FilterPollForRegConversion

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterPollForRegConversion (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Timeout)**
Function description	This function allows to poll for the end of regular conversion.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **Timeout:** : Timeout value in milliseconds.
Return values	• **HAL:** status
Notes	• This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterPollForInjConversion

Function name	**HAL_StatusTypeDef HAL_DFSDM_FilterPollForInjConversion (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Timeout)**
Function description	This function allows to poll for the end of injected conversion.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **Timeout:** : Timeout value in milliseconds.
Return values	• **HAL:** status
Notes	• This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterRegConvCpltCallback

Function name	**void HAL_DFSDM_FilterRegConvCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	Regular conversion complete callback.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **None:**
Notes	• In interrupt mode, user has to read conversion value in this function using HAL_DFSDM_FilterGetRegularValue.

HAL_DFSDM_FilterRegConvHalfCpltCallback

Function name	**void HAL_DFSDM_FilterRegConvHalfCpltCallback**

	(DFSDM_Filter_HandleTypeDef * hdfsdm_filter)
Function description	Half regular conversion complete callback.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **None:**

HAL_DFSDM_FilterInjConvCpltCallback

Function name	**void HAL_DFSDM_FilterInjConvCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	Injected conversion complete callback.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **None:**
Notes	• In interrupt mode, user has to read conversion value in this function using HAL_DFSDM_FilterGetInjectedValue.

HAL_DFSDM_FilterInjConvHalfCpltCallback

Function name	**void HAL_DFSDM_FilterInjConvHalfCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	Half injected conversion complete callback.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **None:**

HAL_DFSDM_FilterAwdCallback

Function name	**void HAL_DFSDM_FilterAwdCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Channel, uint32_t Threshold)**
Function description	Filter analog watchdog callback.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle. • **Channel:** : Corresponding channel. • **Threshold:** : Low or high threshold has been reached.
Return values	• **None:**

HAL_DFSDM_FilterErrorCallback

Function name	**void HAL_DFSDM_FilterErrorCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	Error callback.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **None:**

HAL_DFSDM_FilterGetState

Function name	**HAL_DFSDM_Filter_StateTypeDef**

	HAL_DFSDM_FilterGetState (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)
Function description	This function allows to get the current DFSDM filter handle state.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **DFSDM:** filter state.

HAL_DFSDM_FilterGetError

Function name	**uint32_t HAL_DFSDM_FilterGetError (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)**
Function description	This function allows to get the current DFSDM filter error.
Parameters	• **hdfsdm_filter:** : DFSDM filter handle.
Return values	• **DFSDM:** filter error code.

17.3 DFSDM Firmware driver defines

17.3.1 DFSDM

DFSDM analog watchdog threshold

DFSDM_AWD_HIGH_THRESHOLD	Analog watchdog high threshold
DFSDM_AWD_LOW_THRESHOLD	Analog watchdog low threshold

DFSDM break signals

DFSDM_NO_BREAK_SIGNAL	No break signal
DFSDM_BREAK_SIGNAL_0	Break signal 0
DFSDM_BREAK_SIGNAL_1	Break signal 1
DFSDM_BREAK_SIGNAL_2	Break signal 2
DFSDM_BREAK_SIGNAL_3	Break signal 3

DFSDM channel analog watchdog filter order

DFSDM_CHANNEL_FASTSINC_ORDER	FastSinc filter type
DFSDM_CHANNEL_SINC1_ORDER	Sinc 1 filter type
DFSDM_CHANNEL_SINC2_ORDER	Sinc 2 filter type
DFSDM_CHANNEL_SINC3_ORDER	Sinc 3 filter type

DFSDM channel input data packing

DFSDM_CHANNEL_STANDARD_MODE	Standard data packing mode
DFSDM_CHANNEL_INTERLEAVED_MODE	Interleaved data packing mode
DFSDM_CHANNEL_DUAL_MODE	Dual data packing mode

DFSDM channel input multiplexer

DFSDM_CHANNEL_EXTERNAL_INPUTS	Data are taken from external inputs
DFSDM_CHANNEL_ADC_OUTPUT	Data are taken from ADC output
DFSDM_CHANNEL_INTERNAL_REGISTER	Data are taken from internal register

DFSDM channel input pins

DFSDM_CHANNEL_SAME_CHANNEL_PINS	Input from pins on same channel
DFSDM_CHANNEL_FOLLOWING_CHANNEL_PINS	Input from pins on following channel

DFSDM channel output clock selection

DFSDM_CHANNEL_OUTPUT_CLOCK_SYSTEM	Source for ouput clock is system clock
DFSDM_CHANNEL_OUTPUT_CLOCK_AUDIO	Source for ouput clock is audio clock

DFSDM Channel Selection

DFSDM_CHANNEL_0

DFSDM_CHANNEL_1

DFSDM_CHANNEL_2

DFSDM_CHANNEL_3

DFSDM_CHANNEL_4

DFSDM_CHANNEL_5

DFSDM_CHANNEL_6

DFSDM_CHANNEL_7

DFSDM channel serial interface type

DFSDM_CHANNEL_SPI_RISING	SPI with rising edge
DFSDM_CHANNEL_SPI_FALLING	SPI with falling edge
DFSDM_CHANNEL_MANCHESTER_RISING	Manchester with rising edge
DFSDM_CHANNEL_MANCHESTER_FALLING	Manchester with falling edge

DFSDM channel SPI clock selection

DFSDM_CHANNEL_SPI_CLOCK_EXTERNAL	External SPI clock
DFSDM_CHANNEL_SPI_CLOCK_INTERNAL	Internal SPI clock
DFSDM_CHANNEL_SPI_CLOCK_INTERNAL_DIV2_FALLING	Internal SPI clock divided by 2, falling edge
DFSDM_CHANNEL_SPI_CLOCK_INTERNAL_DIV2_RISING	Internal SPI clock divided by 2, rising edge

DFSDM Continuous Mode

DFSDM_CONTINUOUS_CONV_OFF	Conversion are not continuous
DFSDM_CONTINUOUS_CONV_ON	Conversion are continuous

DFSDM Exported Macros

__HAL_DFSDM_CHANNEL_RESET_HANDLE_STATE

Description:

- Reset DFSDM channel handle state.

Parameters:

- __HANDLE__: DFSDM channel handle.

	Return value: • None
__HAL_DFSDM_FILTER_RESET_HANDLE_STATE	**Description:** • Reset DFSDM filter handle state. **Parameters:** • __HANDLE__: DFSDM filter handle. **Return value:** • None

DFSDM filter analog watchdog data source

DFSDM_FILTER_AWD_FILTER_DATA	From digital filter
DFSDM_FILTER_AWD_CHANNEL_DATA	From analog watchdog channel

DFSDM filter error code

DFSDM_FILTER_ERROR_NONE	No error
DFSDM_FILTER_ERROR_REGULAR_OVERRUN	Overrun occurs during regular conversion
DFSDM_FILTER_ERROR_INJECTED_OVERRUN	Overrun occurs during injected conversion
DFSDM_FILTER_ERROR_DMA	DMA error occurs

DFSDM filter external trigger

DFSDM_FILTER_EXT_TRIG_TIM1_TRGO	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_TIM1_TRGO2	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_TIM8_TRGO	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_TIM8_TRGO2	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_TIM3_TRGO	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_TIM4_TRGO	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_TIM16_OC1	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_TIM6_TRGO	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_TIM7_TRGO	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_EXTI11	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_EXTI15	For all DFSDM filters
DFSDM_FILTER_EXT_TRIG_LPTIM1_OUT	For all DFSDM filters

DFSDM filter external trigger edge

DFSDM_FILTER_EXT_TRIG_RISING_EDGE	External rising edge
DFSDM_FILTER_EXT_TRIG_FALLING_EDGE	External falling edge
DFSDM_FILTER_EXT_TRIG_BOTH_EDGES	External rising and falling edges

DFSDM filter sinc order

DFSDM_FILTER_FASTSINC_ORDER	FastSinc filter type
DFSDM_FILTER_SINC1_ORDER	Sinc 1 filter type
DFSDM_FILTER_SINC2_ORDER	Sinc 2 filter type
DFSDM_FILTER_SINC3_ORDER	Sinc 3 filter type
DFSDM_FILTER_SINC4_ORDER	Sinc 4 filter type
DFSDM_FILTER_SINC5_ORDER	Sinc 5 filter type

DFSDM filter conversion trigger

DFSDM_FILTER_SW_TRIGGER	Software trigger
DFSDM_FILTER_SYNC_TRIGGER	Synchronous with DFSDM_FLT0
DFSDM_FILTER_EXT_TRIGGER	External trigger (only for injected conversion)

18 HAL DFSDM Extension Driver

18.1 DFSDMEx Firmware driver API description

18.1.1 Extended channel operation functions

This section provides functions allowing to:

- Set and get value of pulses skipping on channel

This section contains the following APIs:

- *HAL_DFDSMEx_ChannelSetPulsesSkipping()*
- *HAL_DFDSMEx_ChannelGetPulsesSkipping()*

18.1.2 Detailed description of functions

HAL_DFDSMEx_ChannelSetPulsesSkipping

Function name	**HAL_StatusTypeDef HAL_DFDSMEx_ChannelSetPulsesSkipping (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t PulsesValue)**
Function description	Set value of pulses skipping.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle. • **PulsesValue:** Value of pulses to be skipped. This parameter must be a number between Min_Data = 0 and Max_Data = 63.
Return values	• **HAL:** status.

HAL_DFDSMEx_ChannelGetPulsesSkipping

Function name	**HAL_StatusTypeDef HAL_DFDSMEx_ChannelGetPulsesSkipping (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t * PulsesValue)**
Function description	Get value of pulses skipping.
Parameters	• **hdfsdm_channel:** : DFSDM channel handle. • **PulsesValue:** Value of pulses to be skipped.
Return values	• **HAL:** status.

19 HAL DMA2D Generic Driver

19.1 DMA2D Firmware driver registers structures

19.1.1 DMA2D_ColorTypeDef

Data Fields

- ***uint32_t Blue***
- ***uint32_t Green***
- ***uint32_t Red***

Field Documentation

- ***uint32_t DMA2D_ColorTypeDef::Blue***
  Configures the blue value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t DMA2D_ColorTypeDef::Green***
  Configures the green value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t DMA2D_ColorTypeDef::Red***
  Configures the red value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

19.1.2 DMA2D_CLUTCfgTypeDef

Data Fields

- ***uint32_t * pCLUT***
- ***uint32_t CLUTColorMode***
- ***uint32_t Size***

Field Documentation

- ***uint32_t* DMA2D_CLUTCfgTypeDef::pCLUT***
  Configures the DMA2D CLUT memory address.
- ***uint32_t DMA2D_CLUTCfgTypeDef::CLUTColorMode***
  Configures the DMA2D CLUT color mode. This parameter can be one value of ***DMA2D_CLUT_CM***.
- ***uint32_t DMA2D_CLUTCfgTypeDef::Size***
  Configures the DMA2D CLUT size. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

19.1.3 DMA2D_InitTypeDef

Data Fields

- ***uint32_t Mode***
- ***uint32_t ColorMode***
- ***uint32_t OutputOffset***
- ***uint32_t AlphaInverted***
- ***uint32_t RedBlueSwap***
- ***uint32_t BytesSwap***
- ***uint32_t LineOffsetMode***

Field Documentation

- ***uint32_t DMA2D_InitTypeDef::Mode***
  Configures the DMA2D transfer mode. This parameter can be one value of ***DMA2D_Mode***.
- ***uint32_t DMA2D_InitTypeDef::ColorMode***
  Configures the color format of the output image. This parameter can be one value of ***DMA2D_Output_Color_Mode***.
- ***uint32_t DMA2D_InitTypeDef::OutputOffset***
  Specifies the Offset value. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- ***uint32_t DMA2D_InitTypeDef::AlphaInverted***
  Select regular or inverted alpha value for the output pixel format converter. This parameter can be one value of ***DMA2D_Alpha_Inverted***.
- ***uint32_t DMA2D_InitTypeDef::RedBlueSwap***
  Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR) for the output pixel format converter. This parameter can be one value of ***DMA2D_RB_Swap***.
- ***uint32_t DMA2D_InitTypeDef::BytesSwap***
  Select byte regular mode or bytes swap mode (two by two). This parameter can be one value of ***DMA2D_Bytes_Swap***.
- ***uint32_t DMA2D_InitTypeDef::LineOffsetMode***
  Configures how is expressed the line offset for the foreground, background and output. This parameter can be one value of ***DMA2D_Line_Offset_Mode***.

19.1.4 DMA2D_LayerCfgTypeDef

Data Fields

- ***uint32_t InputOffset***
- ***uint32_t InputColorMode***
- ***uint32_t AlphaMode***
- ***uint32_t InputAlpha***
- ***uint32_t AlphaInverted***
- ***uint32_t RedBlueSwap***

Field Documentation

- ***uint32_t DMA2D_LayerCfgTypeDef::InputOffset***
  Configures the DMA2D foreground or background offset. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- ***uint32_t DMA2D_LayerCfgTypeDef::InputColorMode***
  Configures the DMA2D foreground or background color mode. This parameter can be one value of ***DMA2D_Input_Color_Mode***.
- ***uint32_t DMA2D_LayerCfgTypeDef::AlphaMode***
  Configures the DMA2D foreground or background alpha mode. This parameter can be one value of ***DMA2D_Alpha_Mode***.
- ***uint32_t DMA2D_LayerCfgTypeDef::InputAlpha***
  Specifies the DMA2D foreground or background alpha value and color value in case of A8 or A4 color mode. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF except for the color modes detailed below.
  Note: In case of A8 or A4 color mode (ARGB), this parameter must be a number between Min_Data = 0x00000000 and Max_Data = 0xFFFFFFFF whereInputAlpha[24:31] is the alpha value ALPHA[0:7]InputAlpha[16:23] is the red value RED[0:7]InputAlpha[8:15] is the green value GREEN[0:7]InputAlpha[0:7] is the blue value BLUE[0:7].
- ***uint32_t DMA2D_LayerCfgTypeDef::AlphaInverted***
  Select regular or inverted alpha value. This parameter can be one value of ***DMA2D_Alpha_Inverted***.

- ***uint32_t DMA2D_LayerCfgTypeDef::RedBlueSwap***
  Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR). This parameter can be one value of ***DMA2D_RB_Swap***.

19.1.5 __DMA2D_HandleTypeDef

Data Fields

- ***DMA2D_TypeDef * Instance***
- ***DMA2D_InitTypeDef Init***
- ***void(* XferCpltCallback***
- ***void(* XferErrorCallback***
- ***DMA2D_LayerCfgTypeDef LayerCfg***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DMA2D_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***DMA2D_TypeDef* __DMA2D_HandleTypeDef::Instance***
  DMA2D register base address.
- ***DMA2D_InitTypeDef __DMA2D_HandleTypeDef::Init***
  DMA2D communication parameters.
- ***void(* __DMA2D_HandleTypeDef::XferCpltCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D transfer complete callback.
- ***void(* __DMA2D_HandleTypeDef::XferErrorCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D transfer error callback.
- ***DMA2D_LayerCfgTypeDef __DMA2D_HandleTypeDef::LayerCfg[MAX_DMA2D_LAYER]***
  DMA2D Layers parameters
- ***HAL_LockTypeDef __DMA2D_HandleTypeDef::Lock***
  DMA2D lock.
- ***__IO HAL_DMA2D_StateTypeDef __DMA2D_HandleTypeDef::State***
  DMA2D transfer state.
- ***__IO uint32_t __DMA2D_HandleTypeDef::ErrorCode***
  DMA2D error code.

19.2 DMA2D Firmware driver API description

19.2.1 How to use this driver

1. Program the required configuration through the following parameters: the transfer mode, the output color mode and the output offset using HAL_DMA2D_Init() function.
2. Program the required configuration through the following parameters: the input color mode, the input color, the input alpha value, the alpha mode, the red/blue swap mode, the inverted alpha mode and the input offset using HAL_DMA2D_ConfigLayer() function for foreground or/and background layer.

Polling mode IO operation

1. Configure pdata parameter (explained hereafter), destination and data length and enable the transfer using HAL_DMA2D_Start().
2. Wait for end of transfer using HAL_DMA2D_PollForTransfer(), at this stage user can specify the value of timeout according to his end application.

Interrupt mode IO operation

1. Configure pdata parameter, destination and data length and enable the transfer using HAL_DMA2D_Start_IT().
2. Use HAL_DMA2D_IRQHandler() called under DMA2D_IRQHandler() interrupt subroutine.
3. At the end of data transfer HAL_DMA2D_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback (member of DMA2D handle structure).
4. In case of error, the HAL_DMA2D_IRQHandler() function calls the callback XferErrorCallback. In Register-to-Memory transfer mode, pdata parameter is the register color, in Memory-to-memory or Memory-to-Memory with pixel format conversion pdata is the source address. Configure the foreground source address, the background source address, the destination and data length then Enable the transfer using HAL_DMA2D_BlendingStart() in polling mode and HAL_DMA2D_BlendingStart_IT() in interrupt mode. HAL_DMA2D_BlendingStart() and HAL_DMA2D_BlendingStart_IT() functions are used if the memory to memory with blending transfer mode is selected.
5. Optionally, configure and enable the CLUT using HAL_DMA2D_CLUTLoad() in polling mode or HAL_DMA2D_CLUTLoad_IT() in interrupt mode.
6. Optionally, configure the line watermark in using the API HAL_DMA2D_ProgramLineEvent().
7. Optionally, configure the dead time value in the AHB clock cycle inserted between two consecutive accesses on the AHB master port in using the API HAL_DMA2D_ConfigDeadTime() and enable/disable the functionality with the APIs HAL_DMA2D_EnableDeadTime() or HAL_DMA2D_DisableDeadTime().
8. The transfer can be suspended, resumed and aborted using the following functions: HAL_DMA2D_Suspend(), HAL_DMA2D_Resume(), HAL_DMA2D_Abort().
9. The CLUT loading can be suspended, resumed and aborted using the following functions: HAL_DMA2D_CLUTLoading_Suspend(), HAL_DMA2D_CLUTLoading_Resume(), HAL_DMA2D_CLUTLoading_Abort().
10. To control the DMA2D state, use the following function: HAL_DMA2D_GetState().
11. To read the DMA2D error code, use the following function: HAL_DMA2D_GetError().

DMA2D HAL driver macros list

Below the list of most used macros in DMA2D HAL driver:

- __HAL_DMA2D_ENABLE: Enable the DMA2D peripheral.
- __HAL_DMA2D_GET_FLAG: Get the DMA2D pending flags.
- __HAL_DMA2D_CLEAR_FLAG: Clear the DMA2D pending flags.
- __HAL_DMA2D_ENABLE_IT: Enable the specified DMA2D interrupts.
- __HAL_DMA2D_DISABLE_IT: Disable the specified DMA2D interrupts.
- __HAL_DMA2D_GET_IT_SOURCE: Check whether the specified DMA2D interrupt is enabled or not.

You can refer to the DMA2D HAL driver header file for more useful macros

19.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DMA2D
- De-initialize the DMA2D

This section contains the following APIs:

- ***HAL_DMA2D_Init()***
- ***HAL_DMA2D_DeInit()***
- ***HAL_DMA2D_MspInit()***
- ***HAL_DMA2D_MspDeInit()***

19.2.3 IO operation functions

This section provides functions allowing to:

- Configure the pdata, destination address and data size then start the DMA2D transfer.
- Configure the source for foreground and background, destination address and data size then start a MultiBuffer DMA2D transfer.
- Configure the pdata, destination address and data size then start the DMA2D transfer with interrupt.
- Configure the source for foreground and background, destination address and data size then start a MultiBuffer DMA2D transfer with interrupt.
- Abort DMA2D transfer.
- Suspend DMA2D transfer.
- Resume DMA2D transfer.
- Enable CLUT transfer.
- Configure CLUT loading then start transfer in polling mode.
- Configure CLUT loading then start transfer in interrupt mode.
- Abort DMA2D CLUT loading.
- Suspend DMA2D CLUT loading.
- Resume DMA2D CLUT loading.
- Poll for transfer complete.
- handle DMA2D interrupt request.
- Transfer watermark callback.
- CLUT Transfer Complete callback.

This section contains the following APIs:

- ***HAL_DMA2D_Start()***
- ***HAL_DMA2D_Start_IT()***
- ***HAL_DMA2D_BlendingStart()***
- ***HAL_DMA2D_BlendingStart_IT()***
- ***HAL_DMA2D_Abort()***
- ***HAL_DMA2D_Suspend()***
- ***HAL_DMA2D_Resume()***
- ***HAL_DMA2D_EnableCLUT()***
- ***HAL_DMA2D_CLUTLoad()***
- ***HAL_DMA2D_CLUTLoad_IT()***
- ***HAL_DMA2D_CLUTLoading_Abort()***
- ***HAL_DMA2D_CLUTLoading_Suspend()***
- ***HAL_DMA2D_CLUTLoading_Resume()***
- ***HAL_DMA2D_PollForTransfer()***
- ***HAL_DMA2D_IRQHandler()***
- ***HAL_DMA2D_LineEventCallback()***
- ***HAL_DMA2D_CLUTLoadingCpltCallback()***

19.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the DMA2D foreground or background layer parameters.
- Configure the DMA2D CLUT transfer.
- Configure the line watermark
- Configure the dead time value.
- Enable or disable the dead time value functionality.

This section contains the following APIs:

- ***HAL_DMA2D_ConfigLayer()***
- ***HAL_DMA2D_ConfigCLUT()***
- ***HAL_DMA2D_ProgramLineEvent()***
- ***HAL_DMA2D_EnableDeadTime()***
- ***HAL_DMA2D_DisableDeadTime()***
- ***HAL_DMA2D_ConfigDeadTime()***

19.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to:

- Get the DMA2D state
- Get the DMA2D error code

This section contains the following APIs:

- ***HAL_DMA2D_GetState()***
- ***HAL_DMA2D_GetError()***

19.2.6 Detailed description of functions

HAL_DMA2D_Init

Function name	**HAL_StatusTypeDef HAL_DMA2D_Init (DMA2D_HandleTypeDef * hdma2d)**
Function description	Initialize the DMA2D according to the specified parameters in the DMA2D_InitTypeDef and create the associated handle.
Parameters	• **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **HAL:** status

HAL_DMA2D_DeInit

Function name	**HAL_StatusTypeDef HAL_DMA2D_DeInit (DMA2D_HandleTypeDef * hdma2d)**
Function description	Deinitializes the DMA2D peripheral registers to their default reset values.
Parameters	• **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **None:**

HAL_DMA2D_MspInit

Function name	**void HAL_DMA2D_MspInit (DMA2D_HandleTypeDef * hdma2d)**

Function description	Initializes the DMA2D MSP.
Parameters	• **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **None:**

HAL_DMA2D_MspDeInit

Function name	**void HAL_DMA2D_MspDeInit (DMA2D_HandleTypeDef * hdma2d)**
Function description	DeInitializes the DMA2D MSP.
Parameters	• **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **None:**

HAL_DMA2D_Start

Function name	**HAL_StatusTypeDef HAL_DMA2D_Start (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)**
Function description	Start the DMA2D Transfer.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **pdata:** Configure the source memory Buffer address if Memory-to-Memory or Memory-to-Memory with pixel format conversion mode is selected, or configure the color value if Register-to-Memory mode is selected. • **DstAddress:** The destination memory Buffer address. • **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line). • **Height:** The height of data to be transferred from source to destination (expressed in number of lines).
Return values	• **HAL:** status

HAL_DMA2D_BlendingStart

Function name	**HAL_StatusTypeDef HAL_DMA2D_BlendingStart (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)**
Function description	Start the multi-source DMA2D Transfer.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **SrcAddress1:** The source memory Buffer address for the foreground layer. • **SrcAddress2:** The source memory Buffer address for the background layer. • **DstAddress:** The destination memory Buffer address. • **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).

	• **Height:** The height of data to be transferred from source to destination (expressed in number of lines).
Return values	• **HAL:** status

HAL_DMA2D_Start_IT

Function name	**HAL_StatusTypeDef HAL_DMA2D_Start_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)**
Function description	Start the DMA2D Transfer with interrupt enabled.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **pdata:** Configure the source memory Buffer address if the Memory-to-Memory or Memory-to-Memory with pixel format conversion mode is selected, or configure the color value if Register-to-Memory mode is selected. • **DstAddress:** The destination memory Buffer address. • **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line). • **Height:** The height of data to be transferred from source to destination (expressed in number of lines).
Return values	• **HAL:** status

HAL_DMA2D_BlendingStart_IT

Function name	**HAL_StatusTypeDef HAL_DMA2D_BlendingStart_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)**
Function description	Start the multi-source DMA2D Transfer with interrupt enabled.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **SrcAddress1:** The source memory Buffer address for the foreground layer. • **SrcAddress2:** The source memory Buffer address for the background layer. • **DstAddress:** The destination memory Buffer address. • **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line). • **Height:** The height of data to be transferred from source to destination (expressed in number of lines).
Return values	• **HAL:** status

HAL_DMA2D_Suspend

Function name	**HAL_StatusTypeDef HAL_DMA2D_Suspend (DMA2D_HandleTypeDef * hdma2d)**
Function description	Suspend the DMA2D Transfer.
Parameters	• **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that

contains the configuration information for the DMA2D.

Return values	• **HAL:** status

HAL_DMA2D_Resume

Function name	**HAL_StatusTypeDef HAL_DMA2D_Resume (DMA2D_HandleTypeDef * hdma2d)**
Function description	Resume the DMA2D Transfer.
Parameters	• **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **HAL:** status

HAL_DMA2D_Abort

Function name	**HAL_StatusTypeDef HAL_DMA2D_Abort (DMA2D_HandleTypeDef * hdma2d)**
Function description	Abort the DMA2D Transfer.
Parameters	• **hdma2d:** : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **HAL:** status

HAL_DMA2D_EnableCLUT

Function name	**HAL_StatusTypeDef HAL_DMA2D_EnableCLUT (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)**
Function description	Enable the DMA2D CLUT Transfer.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)
Return values	• **HAL:** status

HAL_DMA2D_CLUTLoad

Function name	**HAL_StatusTypeDef HAL_DMA2D_CLUTLoad (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)**
Function description	Start DMA2D CLUT Loading.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table. • **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)
Return values	• **HAL:** status

Notes	• Invoking this API is similar to calling HAL_DMA2D_ConfigCLUT() then HAL_DMA2D_EnableCLUT().

HAL_DMA2D_CLUTLoad_IT

Function name	**HAL_StatusTypeDef HAL_DMA2D_CLUTLoad_IT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)**
Function description	Start DMA2D CLUT Loading with interrupt enabled.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table. • **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)
Return values	• **HAL:** status

HAL_DMA2D_CLUTLoading_Abort

Function name	**HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Abort (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)**
Function description	Abort the DMA2D CLUT loading.
Parameters	• **hdma2d:** : Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)
Return values	• **HAL:** status

HAL_DMA2D_CLUTLoading_Suspend

Function name	**HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Suspend (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)**
Function description	Suspend the DMA2D CLUT loading.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)
Return values	• **HAL:** status

HAL_DMA2D_CLUTLoading_Resume

Function name	**HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Resume (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)**
Function description	Resume the DMA2D CLUT loading.
Parameters	• **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

	• **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)
Return values	• **HAL:** status

HAL_DMA2D_PollForTransfer

Function name	**HAL_StatusTypeDef HAL_DMA2D_PollForTransfer (DMA2D_HandleTypeDef * hdma2d, uint32_t Timeout)**
Function description	Polling for transfer complete or CLUT loading.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_DMA2D_IRQHandler

Function name	**void HAL_DMA2D_IRQHandler (DMA2D_HandleTypeDef * hdma2d)**
Function description	Handle DMA2D interrupt request.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **HAL:** status

HAL_DMA2D_LineEventCallback

Function name	**void HAL_DMA2D_LineEventCallback (DMA2D_HandleTypeDef * hdma2d)**
Function description	Transfer watermark callback.
Parameters	• **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **None:**

HAL_DMA2D_CLUTLoadingCpltCallback

Function name	**void HAL_DMA2D_CLUTLoadingCpltCallback (DMA2D_HandleTypeDef * hdma2d)**
Function description	CLUT Transfer Complete callback.
Parameters	• **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **None:**

HAL_DMA2D_ConfigLayer

Function name	**HAL_StatusTypeDef HAL_DMA2D_ConfigLayer (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)**
Function description	Configure the DMA2D Layer according to the specified parameters

	in the DMA2D_HandleTypeDef.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)
Return values	• **HAL:** status

HAL_DMA2D_ConfigCLUT

Function name	**HAL_StatusTypeDef HAL_DMA2D_ConfigCLUT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)**
Function description	Configure the DMA2D CLUT Transfer.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table. • **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)
Return values	• **HAL:** status

HAL_DMA2D_ProgramLineEvent

Function name	**HAL_StatusTypeDef HAL_DMA2D_ProgramLineEvent (DMA2D_HandleTypeDef * hdma2d, uint32_t Line)**
Function description	Configure the line watermark.
Parameters	• **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **Line:** Line Watermark configuration (maximum 16-bit long value expected).
Return values	• **HAL:** status
Notes	• HAL_DMA2D_ProgramLineEvent() API enables the transfer watermark interrupt. • The transfer watermark interrupt is disabled once it has occurred.

HAL_DMA2D_EnableDeadTime

Function name	**HAL_StatusTypeDef HAL_DMA2D_EnableDeadTime (DMA2D_HandleTypeDef * hdma2d)**
Function description	Enable DMA2D dead time feature.
Parameters	• **hdma2d:** DMA2D handle.
Return values	• **HAL:** status

HAL_DMA2D_DisableDeadTime

Function name	**HAL_StatusTypeDef HAL_DMA2D_DisableDeadTime (DMA2D_HandleTypeDef * hdma2d)**
Function description	Disable DMA2D dead time feature.
Parameters	• **hdma2d:** DMA2D handle.
Return values	• **HAL:** status

HAL_DMA2D_ConfigDeadTime

Function name	**HAL_StatusTypeDef HAL_DMA2D_ConfigDeadTime (DMA2D_HandleTypeDef * hdma2d, uint8_t DeadTime)**
Function description	Configure dead time.
Parameters	• **hdma2d:** DMA2D handle. • **DeadTime:** dead time value.
Return values	• **HAL:** status
Notes	• The dead time value represents the guaranteed minimum number of cycles between two consecutive transactions on the AHB bus.

HAL_DMA2D_GetState

Function name	**HAL_DMA2D_StateTypeDef HAL_DMA2D_GetState (DMA2D_HandleTypeDef * hdma2d)**
Function description	Return the DMA2D state.
Parameters	• **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **HAL:** state

HAL_DMA2D_GetError

Function name	**uint32_t HAL_DMA2D_GetError (DMA2D_HandleTypeDef * hdma2d)**
Function description	Return the DMA2D error code.
Parameters	• **hdma2d:** : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for DMA2D.
Return values	• **DMA2D:** Error Code

19.3 DMA2D Firmware driver defines

19.3.1 DMA2D

DMA2D API Aliases

HAL_DMA2D_DisableCLUT	Aliased to HAL_DMA2D_CLUTLoading_Abort for compatibility with legacy code

DMA2D Alpha Inversion

DMA2D_REGULAR_ALPHA	No modification of the alpha channel value
DMA2D_INVERTED_ALPHA	Invert the alpha channel value

DMA2D Alpha Mode

DMA2D_NO_MODIF_ALPHA	No modification of the alpha channel value
DMA2D_REPLACE_ALPHA	Replace original alpha channel value by programmed alpha value
DMA2D_COMBINE_ALPHA	Replace original alpha channel value by programmed alpha value with original alpha channel value

DMA2D Bytes Swap

DMA2D_BYTES_REGULAR	Bytes in regular order in output FIFO
DMA2D_BYTES_SWAP	Bytes are swapped two by two in output FIFO

DMA2D CLUT Color Mode

DMA2D_CCM_ARGB8888	ARGB8888 DMA2D CLUT color mode
DMA2D_CCM_RGB888	RGB888 DMA2D CLUT color mode

DMA2D CLUT Size

DMA2D_CLUT_SIZE	DMA2D CLUT size

DMA2D Color Value

DMA2D_COLOR_VALUE	Color value mask

DMA2D Error Code

HAL_DMA2D_ERROR_NONE	No error
HAL_DMA2D_ERROR_TE	Transfer error
HAL_DMA2D_ERROR_CE	Configuration error
HAL_DMA2D_ERROR_CAE	CLUT access error
HAL_DMA2D_ERROR_TIMEOUT	Timeout error

DMA2D Exported Macros

__HAL_DMA2D_RESET_HANDLE_STATE

Description:

- Reset DMA2D handle state.

Parameters:

- __HANDLE__: specifies the DMA2D handle.

Return value:

- None

__HAL_DMA2D_ENABLE

Description:

- Enable the DMA2D.

Parameters:

- __HANDLE__: DMA2D handle

Return value:

- None.

__HAL_DMA2D_GET_FLAG

Description:

- Get the DMA2D pending flags.

Parameters:

- __HANDLE__: DMA2D handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  - DMA2D_FLAG_CE: Configuration error flag
  - DMA2D_FLAG_CTC: CLUT transfer complete flag
  - DMA2D_FLAG_CAE: CLUT access error flag
  - DMA2D_FLAG_TW: Transfer Watermark flag
  - DMA2D_FLAG_TC: Transfer complete flag
  - DMA2D_FLAG_TE: Transfer error flag

Return value:

- The: state of FLAG.

__HAL_DMA2D_CLEAR_FLAG

Description:

- Clear the DMA2D pending flags.

Parameters:

- __HANDLE__: DMA2D handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA2D_FLAG_CE: Configuration error flag
  - DMA2D_FLAG_CTC: CLUT transfer complete flag
  - DMA2D_FLAG_CAE: CLUT access error flag
  - DMA2D_FLAG_TW: Transfer Watermark flag
  - DMA2D_FLAG_TC: Transfer complete flag
  - DMA2D_FLAG_TE: Transfer error flag

Return value:

- None

__HAL_DMA2D_ENABLE_IT

Description:

- Enable the specified DMA2D interrupts.

Parameters:

- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DMA2D_IT_CE: Configuration error interrupt mask
  - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  - DMA2D_IT_CAE: CLUT access error interrupt mask
  - DMA2D_IT_TW: Transfer Watermark interrupt mask
  - DMA2D_IT_TC: Transfer complete interrupt mask
  - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA2D_DISABLE_IT

Description:

- Disable the specified DMA2D interrupts.

Parameters:

- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DMA2D_IT_CE: Configuration error interrupt mask
  - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  - DMA2D_IT_CAE: CLUT access error interrupt mask
  - DMA2D_IT_TW: Transfer Watermark interrupt mask
  - DMA2D_IT_TC: Transfer complete interrupt mask
  - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA2D_GET_IT_SOURCE

Description:

- Check whether the specified DMA2D interrupt source is enabled or not.

Parameters:

- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt source to check. This

parameter can be one of the following values:
- DMA2D_IT_CE: Configuration error interrupt mask
- DMA2D_IT_CTC: CLUT transfer complete interrupt mask
- DMA2D_IT_CAE: CLUT access error interrupt mask
- DMA2D_IT_TW: Transfer Watermark interrupt mask
- DMA2D_IT_TC: Transfer complete interrupt mask
- DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- The: state of INTERRUPT source.

DMA2D Flags

DMA2D_FLAG_CE	Configuration Error Interrupt Flag
DMA2D_FLAG_CTC	CLUT Transfer Complete Interrupt Flag
DMA2D_FLAG_CAE	CLUT Access Error Interrupt Flag
DMA2D_FLAG_TW	Transfer Watermark Interrupt Flag
DMA2D_FLAG_TC	Transfer Complete Interrupt Flag
DMA2D_FLAG_TE	Transfer Error Interrupt Flag

DMA2D Input Color Mode

DMA2D_INPUT_ARGB8888	ARGB8888 color mode
DMA2D_INPUT_RGB888	RGB888 color mode
DMA2D_INPUT_RGB565	RGB565 color mode
DMA2D_INPUT_ARGB1555	ARGB1555 color mode
DMA2D_INPUT_ARGB4444	ARGB4444 color mode
DMA2D_INPUT_L8	L8 color mode
DMA2D_INPUT_AL44	AL44 color mode
DMA2D_INPUT_AL88	AL88 color mode
DMA2D_INPUT_L4	L4 color mode
DMA2D_INPUT_A8	A8 color mode
DMA2D_INPUT_A4	A4 color mode

DMA2D Interrupts

DMA2D_IT_CE	Configuration Error Interrupt
DMA2D_IT_CTC	CLUT Transfer Complete Interrupt
DMA2D_IT_CAE	CLUT Access Error Interrupt
DMA2D_IT_TW	Transfer Watermark Interrupt
DMA2D_IT_TC	Transfer Complete Interrupt

DMA2D_IT_TE	Transfer Error Interrupt

DMA2D Line Offset Mode

DMA2D_LOM_PIXELS	Line offsets expressed in pixels
DMA2D_LOM_BYTES	Line offsets expressed in bytes

DMA2D Maximum Line Watermark

DMA2D_LINE_WATERMARK_MAX	DMA2D maximum line watermark

DMA2D Maximum Number of Layers

DMA2D_MAX_LAYER	DMA2D maximum number of layers

DMA2D Mode

DMA2D_M2M	DMA2D memory to memory transfer mode
DMA2D_M2M_PFC	DMA2D memory to memory with pixel format conversion transfer mode
DMA2D_M2M_BLEND	DMA2D memory to memory with blending transfer mode
DMA2D_R2M	DMA2D register to memory transfer mode
DMA2D_M2M_BLEND_FG	DMA2D memory to memory with blending transfer mode and fixed color FG
DMA2D_M2M_BLEND_BG	DMA2D memory to memory with blending transfer mode and fixed color BG

DMA2D Offset

DMA2D_OFFSET	Line Offset

DMA2D Output Color Mode

DMA2D_OUTPUT_ARGB8888	ARGB8888 DMA2D color mode
DMA2D_OUTPUT_RGB888	RGB888 DMA2D color mode
DMA2D_OUTPUT_RGB565	RGB565 DMA2D color mode
DMA2D_OUTPUT_ARGB1555	ARGB1555 DMA2D color mode
DMA2D_OUTPUT_ARGB4444	ARGB4444 DMA2D color mode

DMA2D Red and Blue Swap

DMA2D_RB_REGULAR	Select regular mode (RGB or ARGB)
DMA2D_RB_SWAP	Select swap mode (BGR or ABGR)

DMA2D Shifts

DMA2D_POSITION_FGPFCCR_CS	Required left shift to set foreground CLUT size
DMA2D_POSITION_BGPFCCR_CS	Required left shift to set background CLUT size
DMA2D_POSITION_FGPFCCR_CCM	Required left shift to set foreground CLUT color mode
DMA2D_POSITION_BGPFCCR_CCM	Required left shift to set background CLUT color mode
DMA2D_POSITION_OPFCCR_AI	Required left shift to set output alpha inversion
DMA2D_POSITION_FGPFCCR_AI	Required left shift to set foreground alpha

	inversion
DMA2D_POSITION_BGPFCCR_AI	Required left shift to set background alpha inversion
DMA2D_POSITION_OPFCCR_RBS	Required left shift to set output Red/Blue swap
DMA2D_POSITION_FGPFCCR_RBS	Required left shift to set foreground Red/Blue swap
DMA2D_POSITION_BGPFCCR_RBS	Required left shift to set background Red/Blue swap
DMA2D_POSITION_AMTCR_DT	Required left shift to set deadtime value
DMA2D_POSITION_FGPFCCR_AM	Required left shift to set foreground alpha mode
DMA2D_POSITION_BGPFCCR_AM	Required left shift to set background alpha mode
DMA2D_POSITION_FGPFCCR_ALPHA	Required left shift to set foreground alpha value
DMA2D_POSITION_BGPFCCR_ALPHA	Required left shift to set background alpha value
DMA2D_POSITION_NLR_PL	Required left shift to set pixels per lines value

DMA2D Size

DMA2D_PIXEL	DMA2D number of pixels per line
DMA2D_LINE	DMA2D number of lines

DMA2D Time Out

DMA2D_TIMEOUT_ABORT	1s
DMA2D_TIMEOUT_SUSPEND	1s

20 HAL DMA Generic Driver

20.1 DMA Firmware driver registers structures

20.1.1 DMA_InitTypeDef

Data Fields

- ***uint32_t Request***
- ***uint32_t Direction***
- ***uint32_t PeriphInc***
- ***uint32_t MemInc***
- ***uint32_t PeriphDataAlignment***
- ***uint32_t MemDataAlignment***
- ***uint32_t Mode***
- ***uint32_t Priority***

Field Documentation

- ***uint32_t DMA_InitTypeDef::Request***
  Specifies the request selected for the specified channel. This parameter can be a value of ***DMA_request***
- ***uint32_t DMA_InitTypeDef::Direction***
  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_Data_transfer_direction***
- ***uint32_t DMA_InitTypeDef::PeriphInc***
  Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of ***DMA_Peripheral_incremented_mode***
- ***uint32_t DMA_InitTypeDef::MemInc***
  Specifies whether the memory address register should be incremented or not. This parameter can be a value of ***DMA_Memory_incremented_mode***
- ***uint32_t DMA_InitTypeDef::PeriphDataAlignment***
  Specifies the Peripheral data width. This parameter can be a value of ***DMA_Peripheral_data_size***
- ***uint32_t DMA_InitTypeDef::MemDataAlignment***
  Specifies the Memory data width. This parameter can be a value of ***DMA_Memory_data_size***
- ***uint32_t DMA_InitTypeDef::Mode***
  Specifies the operation mode of the DMAy Channelx. This parameter can be a value of ***DMA_mode***
  Note: The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel
- ***uint32_t DMA_InitTypeDef::Priority***
  Specifies the software priority for the DMAy Channelx. This parameter can be a value of ***DMA_Priority_level***

20.1.2 __DMA_HandleTypeDef

Data Fields

- ***DMA_Channel_TypeDef * Instance***
- ***DMA_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***

- ***__IO HAL_DMA_StateTypeDef State***
- ***void * Parent***
- ***void(* XferCpltCallback***
- ***void(* XferHalfCpltCallback***
- ***void(* XferErrorCallback***
- ***void(* XferAbortCallback***
- ***__IO uint32_t ErrorCode***
- ***DMA_TypeDef * DmaBaseAddress***
- ***uint32_t ChannelIndex***
- ***DMAMUX_Channel_TypeDef * DMAmuxChannel***
- ***DMAMUX_ChannelStatus_TypeDef * DMAmuxChannelStatus***
- ***uint32_t DMAmuxChannelStatusMask***
- ***DMAMUX_RequestGen_TypeDef * DMAmuxRequestGen***
- ***DMAMUX_RequestGenStatus_TypeDef * DMAmuxRequestGenStatus***
- ***uint32_t DMAmuxRequestGenStatusMask***

Field Documentation

- ***DMA_Channel_TypeDef* __DMA_HandleTypeDef::Instance***
  Register base address
- ***DMA_InitTypeDef __DMA_HandleTypeDef::Init***
  DMA communication parameters
- ***HAL_LockTypeDef __DMA_HandleTypeDef::Lock***
  DMA locking object
- ***__IO HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State***
  DMA transfer state
- ***void* __DMA_HandleTypeDef::Parent***
  Parent object state
- ***void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA Half transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer error callback
- ***void(* __DMA_HandleTypeDef::XferAbortCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer abort callback
- ***__IO uint32_t __DMA_HandleTypeDef::ErrorCode***
  DMA Error code
- ***DMA_TypeDef* __DMA_HandleTypeDef::DmaBaseAddress***
  DMA Channel Base Address
- ***uint32_t __DMA_HandleTypeDef::ChannelIndex***
  DMA Channel Index
- ***DMAMUX_Channel_TypeDef* __DMA_HandleTypeDef::DMAmuxChannel***
  Register base address
- ***DMAMUX_ChannelStatus_TypeDef* __DMA_HandleTypeDef::DMAmuxChannelStatus***
  DMAMUX Channels Status Base Address
- ***uint32_t __DMA_HandleTypeDef::DMAmuxChannelStatusMask***
  DMAMUX Channel Status Mask
- ***DMAMUX_RequestGen_TypeDef* __DMA_HandleTypeDef::DMAmuxRequestGen***
  DMAMUX request generator Base Address

- ***DMAMUX_RequestGenStatus_TypeDef* __DMA_HandleTypeDef::DMAmuxRequestGenStatus***
  DMAMUX request generator Address
- ***uint32_t __DMA_HandleTypeDef::DMAmuxRequestGenStatusMask***
  DMAMUX request generator Status mask

20.2 DMA Firmware driver API description

20.2.1 How to use this driver

1. Enable and configure the peripheral to be connected to the DMA Channel (except for internal SRAM / FLASH memories: no initialization is necessary). Please refer to the Reference manual for connection between peripherals and DMA requests.
2. For a given Channel, program the required configuration through the following parameters: Channel request, Transfer Direction, Source and Destination data formats, Circular or Normal mode, Channel Priority level, Source and Destination Increment mode using HAL_DMA_Init() function. Prior to HAL_DMA_Init the peripheral clock shall be enabled for both DMA & DMAMUX thanks to:
   a. DMA1 or DMA2: __HAL_RCC_DMA1_CLK_ENABLE() or __HAL_RCC_DMA2_CLK_ENABLE() ;
   b. DMAMUX1: __HAL_RCC_DMAMUX1_CLK_ENABLE();
3. Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
4. Use HAL_DMA_Abort() function to abort the current transfer In Memory-to-Memory transfer mode, Circular mode is not allowed.

Polling mode IO operation

- Use HAL_DMA_Start() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred
- Use HAL_DMA_PollForTransfer() to poll for the end of current transfer, in this case a fixed Timeout can be configured by User depending from his application.

Interrupt mode IO operation

- Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
- Use HAL_DMA_Start_IT() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL_DMA_IRQHandler() function is executed and user can add his own function to register callbacks with HAL_DMA_RegisterCallback().

DMA HAL driver macros list

Below the list of macros in DMA HAL driver.

- __HAL_DMA_ENABLE: Enable the specified DMA Channel.
- __HAL_DMA_DISABLE: Disable the specified DMA Channel.
- __HAL_DMA_GET_FLAG: Get the DMA Channel pending flags.
- __HAL_DMA_CLEAR_FLAG: Clear the DMA Channel pending flags.
- __HAL_DMA_ENABLE_IT: Enable the specified DMA Channel interrupts.
- __HAL_DMA_DISABLE_IT: Disable the specified DMA Channel interrupts.
- __HAL_DMA_GET_IT_SOURCE: Check whether the specified DMA Channel interrupt is enabled or not.

You can refer to the DMA HAL driver header file for more useful macros

20.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize the DMA Channel source and destination addresses, incrementation and data sizes, transfer direction, circular/normal mode selection, memory-to-memory mode selection and Channel priority value.

The HAL_DMA_Init() function follows the DMA configuration procedures as described in reference manual.

This section contains the following APIs:

- ***HAL_DMA_Init()***
- ***HAL_DMA_DeInit()***

20.2.3 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt
- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request

This section contains the following APIs:

- ***HAL_DMA_Start()***
- ***HAL_DMA_Start_IT()***
- ***HAL_DMA_Abort()***
- ***HAL_DMA_Abort_IT()***
- ***HAL_DMA_PollForTransfer()***
- ***HAL_DMA_IRQHandler()***
- ***HAL_DMA_RegisterCallback()***
- ***HAL_DMA_UnRegisterCallback()***

20.2.4 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code

This section contains the following APIs:

- ***HAL_DMA_GetState()***
- ***HAL_DMA_GetError()***

20.2.5 Detailed description of functions

HAL_DMA_Init

Function name	**HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)**

Function description	Initialize the DMA according to the specified parameters in the DMA_InitTypeDef and initialize the associated handle.
Parameters	• **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **HAL:** status

HAL_DMA_DeInit

Function name	**HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef * hdma)**
Function description	DeInitialize the DMA peripheral.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **HAL:** status

HAL_DMA_Start

Function name	**HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)**
Function description	Start the DMA Transfer.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **SrcAddress:** The source memory Buffer address • **DstAddress:** The destination memory Buffer address • **DataLength:** The length of data to be transferred from source to destination
Return values	• **HAL:** status

HAL_DMA_Start_IT

Function name	**HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)**
Function description	Start the DMA Transfer with interrupt enabled.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **SrcAddress:** The source memory Buffer address • **DstAddress:** The destination memory Buffer address • **DataLength:** The length of data to be transferred from source to destination
Return values	• **HAL:** status

HAL_DMA_Abort

Function name	**HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)**
Function description	Abort the DMA Transfer.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **HAL:** status

HAL_DMA_Abort_IT

Function name	**HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef * hdma)**
Function description	Aborts the DMA Transfer in Interrupt mode.
Parameters	• **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **HAL:** status

HAL_DMA_PollForTransfer

Function name	**HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, HAL_DMA_LevelCompleteTypeDef CompleteLevel, uint32_t Timeout)**
Function description	Polling for transfer complete.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **CompleteLevel:** Specifies the DMA level complete. • **Timeout:** Timeout duration.
Return values	• **HAL:** status

HAL_DMA_IRQHandler

Function name	**void HAL_DMA_IRQHandler (DMA_HandleTypeDef * hdma)**
Function description	Handle DMA interrupt request.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **None:**

HAL_DMA_RegisterCallback

Function name	**HAL_StatusTypeDef HAL_DMA_RegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID, void(*)(DMA_HandleTypeDef *_hdma) pCallback)**

Function description	Register callbacks.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **CallbackID:** User Callback identifer a HAL_DMA_CallbackIDTypeDef ENUM as parameter. • **pCallback:** pointer to private callbacsk function which has pointer to a DMA_HandleTypeDef structure as parameter.
Return values	• **HAL:** status

HAL_DMA_UnRegisterCallback

Function name	**HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID)**
Function description	UnRegister callbacks.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **CallbackID:** User Callback identifer a HAL_DMA_CallbackIDTypeDef ENUM as parameter.
Return values	• **HAL:** status

HAL_DMA_GetState

Function name	**HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef * hdma)**
Function description	Return the DMA hande state.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **HAL:** state

HAL_DMA_GetError

Function name	**uint32_t HAL_DMA_GetError (DMA_HandleTypeDef * hdma)**
Function description	Return the DMA error code.
Parameters	• **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **DMA:** Error Code

20.3 DMA Firmware driver defines

20.3.1 DMA

DMA Data transfer direction

DMA_PERIPH_TO_MEMORY	Peripheral to memory direction
DMA_MEMORY_TO_PERIPH	Memory to peripheral direction
DMA_MEMORY_TO_MEMORY	Memory to memory direction

DMA Error Code

HAL_DMA_ERROR_NONE	No error
HAL_DMA_ERROR_TE	Transfer error
HAL_DMA_ERROR_NO_XFER	Abort requested with no Xfer ongoing
HAL_DMA_ERROR_TIMEOUT	Timeout error
HAL_DMA_ERROR_NOT_SUPPORTED	Not supported mode
HAL_DMA_ERROR_SYNC	DMAMUX sync overrun error
HAL_DMA_ERROR_REQGEN	DMAMUX request generator overrun error

DMA Exported Macros

__HAL_DMA_RESET_HANDLE_STATE

Description:

- Reset DMA handle state.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_ENABLE

Description:

- Enable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_DISABLE

Description:

- Disable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_GET_TC_FLAG_INDEX

Description:

- Return the current DMA Channel transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer complete flag

index.

__HAL_DMA_GET_HT_FLAG_INDEX

Description:

- Return the current DMA Channel half transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified half transfer complete flag index.

__HAL_DMA_GET_TE_FLAG_INDEX

Description:

- Return the current DMA Channel transfer error flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__HAL_DMA_GET_GI_FLAG_INDEX

Description:

- Return the current DMA Channel Global interrupt flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__HAL_DMA_GET_FLAG

Description:

- Get the DMA Channel pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - DMA_FLAG_TCx: Transfer complete flag
  - DMA_FLAG_HTx: Half transfer complete flag
  - DMA_FLAG_TEx: Transfer error flag
  - DMA_FLAG_GLx: Global interrupt flag Where x can be from 1 to 7 to select the DMA Channel x flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_DMA_CLEAR_FLAG

Description:

- Clear the DMA Channel pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA_FLAG_TCx: Transfer complete flag
  - DMA_FLAG_HTx: Half transfer complete flag
  - DMA_FLAG_TEx: Transfer error flag
  - DMA_FLAG_GLx: Global interrupt flag Where x can be from 1 to 7 to select the DMA Channel x flag.

Return value:

- None

__HAL_DMA_ENABLE_IT

Description:

- Enable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA_DISABLE_IT

Description:

- Disable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt

mask

Return value:

- None

__HAL_DMA_GET_IT_SOURCE

Description:

- Check whether the specified DMA Channel interrupt is enabled or not.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt source to check. This parameter can be one of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt mask

Return value:

- The: state of DMA_IT (SET or RESET).

__HAL_DMA_GET_COUNTER

Description:

- Return the number of remaining data units in the current DMA Channel transfer.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: number of remaining data units in the current DMA Channel transfer.

DMA flag definitions

DMA_FLAG_GL1

DMA_FLAG_TC1

DMA_FLAG_HT1

DMA_FLAG_TE1

DMA_FLAG_GL2

DMA_FLAG_TC2

DMA_FLAG_HT2

DMA_FLAG_TE2

DMA_FLAG_GL3

DMA_FLAG_TC3

DMA_FLAG_HT3

DMA_FLAG_TE3

DMA_FLAG_GL4

DMA_FLAG_TC4

DMA_FLAG_HT4

DMA_FLAG_TE4

DMA_FLAG_GL5

DMA_FLAG_TC5

DMA_FLAG_HT5

DMA_FLAG_TE5

DMA_FLAG_GL6

DMA_FLAG_TC6

DMA_FLAG_HT6

DMA_FLAG_TE6

DMA_FLAG_GL7

DMA_FLAG_TC7

DMA_FLAG_HT7

DMA_FLAG_TE7

TIM DMA Handle Index

TIM_DMA_ID_UPDATE	Index of the DMA handle used for Update DMA requests
TIM_DMA_ID_CC1	Index of the DMA handle used for Capture/Compare 1 DMA requests
TIM_DMA_ID_CC2	Index of the DMA handle used for Capture/Compare 2 DMA requests
TIM_DMA_ID_CC3	Index of the DMA handle used for Capture/Compare 3 DMA requests
TIM_DMA_ID_CC4	Index of the DMA handle used for Capture/Compare 4 DMA requests
TIM_DMA_ID_COMMUTATION	Index of the DMA handle used for Commutation DMA requests
TIM_DMA_ID_TRIGGER	Index of the DMA handle used for Trigger DMA requests

DMA interrupt enable definitions

DMA_IT_TC

DMA_IT_HT

DMA_IT_TE

DMA Memory data size

DMA_MDATAALIGN_BYTE	Memory data alignment: Byte
DMA_MDATAALIGN_HALFWORD	Memory data alignment: HalfWord
DMA_MDATAALIGN_WORD	Memory data alignment: Word

DMA Memory incremented mode

DMA_MINC_ENABLE	Memory increment mode Enable
DMA_MINC_DISABLE	Memory increment mode Disable

DMA mode

DMA_NORMAL	Normal mode
DMA_CIRCULAR	Circular mode

DMA Peripheral data size

DMA_PDATAALIGN_BYTE	Peripheral data alignment: Byte
DMA_PDATAALIGN_HALFWORD	Peripheral data alignment: HalfWord
DMA_PDATAALIGN_WORD	Peripheral data alignment: Word

DMA Peripheral incremented mode

DMA_PINC_ENABLE	Peripheral increment mode Enable
DMA_PINC_DISABLE	Peripheral increment mode Disable

DMA Priority level

DMA_PRIORITY_LOW	Priority level: Low
DMA_PRIORITY_MEDIUM	Priority level: Medium
DMA_PRIORITY_HIGH	Priority level: High
DMA_PRIORITY_VERY_HIGH	Priority level: Very_High

DMA request

DMA_REQUEST_MEM2MEM	memory to memory transfer
DMA_REQUEST_GENERATOR0	DMAMUX1 request generator 0
DMA_REQUEST_GENERATOR1	DMAMUX1 request generator 1
DMA_REQUEST_GENERATOR2	DMAMUX1 request generator 2
DMA_REQUEST_GENERATOR3	DMAMUX1 request generator 3
DMA_REQUEST_ADC1	DMAMUX1 ADC1 request
DMA_REQUEST_DAC1_CH1	DMAMUX1 DAC1 CH1 request
DMA_REQUEST_DAC1_CH2	DMAMUX1 DAC1 CH2 request
DMA_REQUEST_TIM6_UP	DMAMUX1 TIM6 UP request
DMA_REQUEST_TIM7_UP	DMAMUX1 TIM7 UP request
DMA_REQUEST_SPI1_RX	DMAMUX1 SPI1 RX request
DMA_REQUEST_SPI1_TX	DMAMUX1 SPI1 TX request
DMA_REQUEST_SPI2_RX	DMAMUX1 SPI2 RX request
DMA_REQUEST_SPI2_TX	DMAMUX1 SPI2 TX request
DMA_REQUEST_SPI3_RX	DMAMUX1 SPI3 RX request
DMA_REQUEST_SPI3_TX	DMAMUX1 SPI3 TX request
DMA_REQUEST_I2C1_RX	DMAMUX1 I2C1 RX request

DMA_REQUEST_I2C1_TX	DMAMUX1 I2C1 TX request
DMA_REQUEST_I2C2_RX	DMAMUX1 I2C2 RX request
DMA_REQUEST_I2C2_TX	DMAMUX1 I2C2 TX request
DMA_REQUEST_I2C3_RX	DMAMUX1 I2C3 RX request
DMA_REQUEST_I2C3_TX	DMAMUX1 I2C3 TX request
DMA_REQUEST_I2C4_RX	DMAMUX1 I2C4 RX request
DMA_REQUEST_I2C4_TX	DMAMUX1 I2C4 TX request
DMA_REQUEST_USART1_RX	DMAMUX1 USART1 RX request
DMA_REQUEST_USART1_TX	DMAMUX1 USART1 TX request
DMA_REQUEST_USART2_RX	DMAMUX1 USART2 RX request
DMA_REQUEST_USART2_TX	DMAMUX1 USART2 TX request
DMA_REQUEST_USART3_RX	DMAMUX1 USART3 RX request
DMA_REQUEST_USART3_TX	DMAMUX1 USART3 TX request
DMA_REQUEST_UART4_RX	DMAMUX1 UART4 RX request
DMA_REQUEST_UART4_TX	DMAMUX1 UART4 TX request
DMA_REQUEST_UART5_RX	DMAMUX1 UART5 RX request
DMA_REQUEST_UART5_TX	DMAMUX1 UART5 TX request
DMA_REQUEST_LPUART1_RX	DMAMUX1 LP_UART1_RX request
DMA_REQUEST_LPUART1_TX	DMAMUX1 LP_UART1_RX request
DMA_REQUEST_SAI1_A	DMAMUX1 SAI1 A request
DMA_REQUEST_SAI1_B	DMAMUX1 SAI1 B request
DMA_REQUEST_SAI2_A	DMAMUX1 SAI2 A request
DMA_REQUEST_SAI2_B	DMAMUX1 SAI2 B request
DMA_REQUEST_OCTOSPI1	DMAMUX1 OCTOSPI1 request
DMA_REQUEST_OCTOSPI2	DMAMUX1 OCTOSPI2 request
DMA_REQUEST_TIM1_CH1	DMAMUX1 TIM1 CH1 request
DMA_REQUEST_TIM1_CH2	DMAMUX1 TIM1 CH2 request
DMA_REQUEST_TIM1_CH3	DMAMUX1 TIM1 CH3 request
DMA_REQUEST_TIM1_CH4	DMAMUX1 TIM1 CH4 request
DMA_REQUEST_TIM1_UP	DMAMUX1 TIM1 UP request
DMA_REQUEST_TIM1_TRIG	DMAMUX1 TIM1 TRIG request
DMA_REQUEST_TIM1_COM	DMAMUX1 TIM1 COM request
DMA_REQUEST_TIM8_CH1	DMAMUX1 TIM8 CH1 request
DMA_REQUEST_TIM8_CH2	DMAMUX1 TIM8 CH2 request
DMA_REQUEST_TIM8_CH3	DMAMUX1 TIM8 CH3 request
DMA_REQUEST_TIM8_CH4	DMAMUX1 TIM8 CH4 request

DMA_REQUEST_TIM8_UP	DMAMUX1 TIM8 UP request
DMA_REQUEST_TIM8_TRIG	DMAMUX1 TIM8 TRIG request
DMA_REQUEST_TIM8_COM	DMAMUX1 TIM8 COM request
DMA_REQUEST_TIM2_CH1	DMAMUX1 TIM2 CH1 request
DMA_REQUEST_TIM2_CH2	DMAMUX1 TIM2 CH2 request
DMA_REQUEST_TIM2_CH3	DMAMUX1 TIM2 CH3 request
DMA_REQUEST_TIM2_CH4	DMAMUX1 TIM2 CH4 request
DMA_REQUEST_TIM2_UP	DMAMUX1 TIM2 UP request
DMA_REQUEST_TIM3_CH1	DMAMUX1 TIM3 CH1 request
DMA_REQUEST_TIM3_CH2	DMAMUX1 TIM3 CH2 request
DMA_REQUEST_TIM3_CH3	DMAMUX1 TIM3 CH3 request
DMA_REQUEST_TIM3_CH4	DMAMUX1 TIM3 CH4 request
DMA_REQUEST_TIM3_UP	DMAMUX1 TIM3 UP request
DMA_REQUEST_TIM3_TRIG	DMAMUX1 TIM3 TRIG request
DMA_REQUEST_TIM4_CH1	DMAMUX1 TIM4 CH1 request
DMA_REQUEST_TIM4_CH2	DMAMUX1 TIM4 CH2 request
DMA_REQUEST_TIM4_CH3	DMAMUX1 TIM4 CH3 request
DMA_REQUEST_TIM4_CH4	DMAMUX1 TIM4 CH4 request
DMA_REQUEST_TIM4_UP	DMAMUX1 TIM4 UP request
DMA_REQUEST_TIM5_CH1	DMAMUX1 TIM5 CH1 request
DMA_REQUEST_TIM5_CH2	DMAMUX1 TIM5 CH2 request
DMA_REQUEST_TIM5_CH3	DMAMUX1 TIM5 CH3 request
DMA_REQUEST_TIM5_CH4	DMAMUX1 TIM5 CH4 request
DMA_REQUEST_TIM5_UP	DMAMUX1 TIM5 UP request
DMA_REQUEST_TIM5_TRIG	DMAMUX1 TIM5 TRIG request
DMA_REQUEST_TIM15_CH1	DMAMUX1 TIM15 CH1 request
DMA_REQUEST_TIM15_UP	DMAMUX1 TIM15 UP request
DMA_REQUEST_TIM15_TRIG	DMAMUX1 TIM15 TRIG request
DMA_REQUEST_TIM15_COM	DMAMUX1 TIM15 COM request
DMA_REQUEST_TIM16_CH1	DMAMUX1 TIM16 CH1 request
DMA_REQUEST_TIM16_UP	DMAMUX1 TIM16 UP request
DMA_REQUEST_TIM17_CH1	DMAMUX1 TIM17 CH1 request
DMA_REQUEST_TIM17_UP	DMAMUX1 TIM17 UP request
DMA_REQUEST_DFSDM1_FLT0	DMAMUX1 DFSDM1 Filter0 request
DMA_REQUEST_DFSDM1_FLT1	DMAMUX1 DFSDM1 Filter1 request
DMA_REQUEST_DFSDM1_FLT2	DMAMUX1 DFSDM1 Filter2 request

DMA_REQUEST_DFSDM1_FLT3	DMAMUX1 DFSDM1 Filter3 request
DMA_REQUEST_DCMI	DMAMUX1 DCMI request
DMA_REQUEST_AES_IN	DMAMUX1 AES IN request
DMA_REQUEST_AES_OUT	DMAMUX1 AES OUT request
DMA_REQUEST_HASH_IN	DMAMUX1 HASH IN request

21 HAL DMA Extension Driver

21.1 DMAEx Firmware driver registers structures

21.1.1 HAL_DMA_MuxSyncConfigTypeDef

Data Fields

- ***uint32_t SyncSignalID***
- ***uint32_t SyncPolarity***
- ***FunctionalState SyncEnable***
- ***FunctionalState EventEnable***
- ***uint32_t RequestNumber***

Field Documentation

- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::SyncSignalID***
  Specifies the synchronization signal gating the DMA request in periodic mode. This parameter can be a value of ***DMAEx_DMAMUX_SyncSignalID_selection***
- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::SyncPolarity***
  Specifies the polarity of the signal on which the DMA request is synchronized. This parameter can be a value of ***DMAEx_DMAMUX_SyncPolarity_selection***
- ***FunctionalState HAL_DMA_MuxSyncConfigTypeDef::SyncEnable***
  Specifies if the synchronization shall be enabled or disabled This parameter can take the value ENABLE or DISABLE
- ***FunctionalState HAL_DMA_MuxSyncConfigTypeDef::EventEnable***
  Specifies if an event shall be generated once the RequestNumber is reached. This parameter can take the value ENABLE or DISABLE
- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::RequestNumber***
  Specifies the number of DMA request that will be authorized after a sync event This parameter must be a number between Min_Data = 1 and Max_Data = 32

21.1.2 HAL_DMA_MuxRequestGeneratorConfigTypeDef

Data Fields

- ***uint32_t SignalID***
- ***uint32_t Polarity***
- ***uint32_t RequestNumber***

Field Documentation

- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::SignalID***
  Specifies the ID of the signal used for DMAMUX request generator This parameter can be a value of ***DMAEx_DMAMUX_SignalGeneratorID_selection***
- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::Polarity***
  Specifies the polarity of the signal on which the request is generated. This parameter can be a value of ***DMAEx_DMAMUX_RequestGeneneratorPolarity_selection***
- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::RequestNumber***
  Specifies the number of DMA request that will be generated after a signal event This parameter must be a number between Min_Data = 1 and Max_Data = 32

21.2 DMAEx Firmware driver API description

21.2.1 How to use this driver

The DMA Extension HAL driver can be used as follows:

- Configure the DMA_MUX Synchronization Block using HAL_DMAEx_ConfigMuxSync function.
- Configure the DMA_MUX Request Generator Block using HAL_DMAEx_ConfigMuxRequestGenerator function. Functions HAL_DMAEx_EnableMuxRequestGenerator and HAL_DMAEx_DisableMuxRequestGenerator can then be used to respectively enable/disable the request generator.
- To handle the DMAMUX Interrupts, the function HAL_DMAEx_MUX_IRQHandler should be called from the DMAMUX IRQ handler i.e DMAMUX1_OVR_IRQHandler. As only one interrupt line is available for all DMAMUX channels and request generators , HAL_DMAEx_MUX_IRQHandler should be called with, as parameter, the appropriate DMA handle as many as used DMAs in the user project (exception done if a given DMA is not using the DMAMUX SYNC block neither a request generator)

21.2.2 Extended features functions

This section provides functions allowing to:

- Configure the DMAMUX Synchronization Block using HAL_DMAEx_ConfigMuxSync function.
- Configure the DMAMUX Request Generator Block using HAL_DMAEx_ConfigMuxRequestGenerator function. Functions HAL_DMAEx_EnableMuxRequestGenerator and HAL_DMAEx_DisableMuxRequestGenerator can then be used to respectively enable/disable the request generator.

This section contains the following APIs:

- ***HAL_DMAEx_ConfigMuxSync()***
- ***HAL_DMAEx_ConfigMuxRequestGenerator()***
- ***HAL_DMAEx_EnableMuxRequestGenerator()***
- ***HAL_DMAEx_DisableMuxRequestGenerator()***
- ***HAL_DMAEx_MUX_IRQHandler()***

21.2.3 Detailed description of functions

HAL_DMAEx_ConfigMuxRequestGenerator

Function name	**HAL_StatusTypeDef HAL_DMAEx_ConfigMuxRequestGenerator (DMA_HandleTypeDef * hdma, HAL_DMA_MuxRequestGeneratorConfigTypeDef * pRequestGeneratorConfig)**
Function description	Configure the DMAMUX request generator block used by the given DMA channel (instance).
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel. • **pRequestGeneratorConfig:** : pointer to HAL_DMA_MuxRequestGeneratorConfigTypeDef: contains

		the request generator parameters.
Return values	•	**HAL:** status

HAL_DMAEx_EnableMuxRequestGenerator

Function name	**HAL_StatusTypeDef HAL_DMAEx_EnableMuxRequestGenerator (DMA_HandleTypeDef * hdma)**
Function description	Enable the DMAMUX request generator block used by the given DMA channel (instance).
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.
Return values	• **HAL:** status

HAL_DMAEx_DisableMuxRequestGenerator

Function name	**HAL_StatusTypeDef HAL_DMAEx_DisableMuxRequestGenerator (DMA_HandleTypeDef * hdma)**
Function description	Disable the DMAMUX request generator block used by the given DMA channel (instance).
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.
Return values	• **HAL:** status

HAL_DMAEx_ConfigMuxSync

Function name	**HAL_StatusTypeDef HAL_DMAEx_ConfigMuxSync (DMA_HandleTypeDef * hdma, HAL_DMA_MuxSyncConfigTypeDef * pSyncConfig)**
Function description	Configure the DMAMUX synchronization parameters for a given DMA channel (instance).
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel. • **pSyncConfig:** : pointer to HAL_DMA_MuxSyncConfigTypeDef: contains the DMAMUX synchronization parameters
Return values	• **HAL:** status

HAL_DMAEx_MUX_IRQHandler

Function name	**void HAL_DMAEx_MUX_IRQHandler (DMA_HandleTypeDef * hdma)**
Function description	Handles DMAMUX interrupt request.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that

	contains the configuration information for the specified DMA channel.
Return values	• **None:**

21.3 DMAEx Firmware driver defines

21.3.1 DMAEx

DMAMUX RequestGeneneratorPolarity selection

HAL_DMAMUX_REQUEST_GEN_NO_EVENT	block request generator events
HAL_DMAMUX_REQUEST_GEN_RISING	generate request on rising edge events
HAL_DMAMUX_REQUEST_GEN_FALLING	generate request on falling edge events
HAL_DMAMUX_REQUEST_GEN_RISING_FALLING	generate request on rising and falling edge events

DMAMUX SignalGeneratorID selection

HAL_DMAMUX1_REQUEST_GEN_EXTI0	Request generator Signal is EXTI0 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI1	Request generator Signal is EXTI1 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI2	Request generator Signal is EXTI2 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI3	Request generator Signal is EXTI3 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI4	Request generator Signal is EXTI4 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI5	Request generator Signal is EXTI5 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI6	Request generator Signal is EXTI6 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI7	Request generator Signal is EXTI7 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI8	Request generator Signal is EXTI8 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI9	Request generator Signal is EXTI9 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI10	Request generator Signal is EXTI10 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI11	Request generator Signal is EXTI11 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI12	Request generator Signal is EXTI12 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI13	Request generator Signal is

	EXTI13 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI14	Request generator Signal is EXTI14 IT
HAL_DMAMUX1_REQUEST_GEN_EXTI15	Request generator Signal is EXTI15 IT
HAL_DMAMUX1_REQUEST_GEN_DMAMUX1_CH0_EVT	Request generator Signal is DMAMUX1 Channel0 Event
HAL_DMAMUX1_REQUEST_GEN_DMAMUX1_CH1_EVT	Request generator Signal is DMAMUX1 Channel1 Event
HAL_DMAMUX1_REQUEST_GEN_DMAMUX1_CH2_EVT	Request generator Signal is DMAMUX1 Channel2 Event
HAL_DMAMUX1_REQUEST_GEN_DMAMUX1_CH3_EVT	Request generator Signal is DMAMUX1 Channel3 Event
HAL_DMAMUX1_REQUEST_GEN_LPTIM1_OUT	Request generator Signal is LPTIM1 OUT
HAL_DMAMUX1_REQUEST_GEN_LPTIM2_OUT	Request generator Signal is LPTIM2 OUT
HAL_DMAMUX1_REQUEST_GEN_DSI_TE	Request generator Signal is DSI Tearing Effect
HAL_DMAMUX1_REQUEST_GEN_DSI_EOT	Request generator Signal is DSI End of refresh
HAL_DMAMUX1_REQUEST_GEN_DMA2D_EOT	Request generator Signal is DMA2D End of Transfer
HAL_DMAMUX1_REQUEST_GEN_LTDC_IT	Request generator Signal is LTDC IT

DMAMUX SyncPolarity selection

HAL_DMAMUX_SYNC_NO_EVENT	block synchronization events
HAL_DMAMUX_SYNC_RISING	synchronize with rising edge events
HAL_DMAMUX_SYNC_FALLING	synchronize with falling edge events
HAL_DMAMUX_SYNC_RISING_FALLING	synchronize with rising and falling edge events

DMAMUX SyncSignalID selection

HAL_DMAMUX1_SYNC_EXTI0	Synchronization Signal is EXTI0 IT
HAL_DMAMUX1_SYNC_EXTI1	Synchronization Signal is EXTI1 IT
HAL_DMAMUX1_SYNC_EXTI2	Synchronization Signal is EXTI2 IT
HAL_DMAMUX1_SYNC_EXTI3	Synchronization Signal is EXTI3 IT
HAL_DMAMUX1_SYNC_EXTI4	Synchronization Signal is EXTI4 IT
HAL_DMAMUX1_SYNC_EXTI5	Synchronization Signal is EXTI5 IT
HAL_DMAMUX1_SYNC_EXTI6	Synchronization Signal is EXTI6 IT
HAL_DMAMUX1_SYNC_EXTI7	Synchronization Signal is EXTI7 IT
HAL_DMAMUX1_SYNC_EXTI8	Synchronization Signal is EXTI8 IT

HAL_DMAMUX1_SYNC_EXTI9	Synchronization Signal is EXTI9 IT
HAL_DMAMUX1_SYNC_EXTI10	Synchronization Signal is EXTI10 IT
HAL_DMAMUX1_SYNC_EXTI11	Synchronization Signal is EXTI11 IT
HAL_DMAMUX1_SYNC_EXTI12	Synchronization Signal is EXTI12 IT
HAL_DMAMUX1_SYNC_EXTI13	Synchronization Signal is EXTI13 IT
HAL_DMAMUX1_SYNC_EXTI14	Synchronization Signal is EXTI14 IT
HAL_DMAMUX1_SYNC_EXTI15	Synchronization Signal is EXTI15 IT
HAL_DMAMUX1_SYNC_DMAMUX1_CH0_EVT	Synchronization Signal is DMAMUX1 Channel0 Event
HAL_DMAMUX1_SYNC_DMAMUX1_CH1_EVT	Synchronization Signal is DMAMUX1 Channel1 Event
HAL_DMAMUX1_SYNC_DMAMUX1_CH2_EVT	Synchronization Signal is DMAMUX1 Channel2 Event
HAL_DMAMUX1_SYNC_DMAMUX1_CH3_EVT	Synchronization Signal is DMAMUX1 Channel3 Event
HAL_DMAMUX1_SYNC_LPTIM1_OUT	Synchronization Signal is LPTIM1 OUT
HAL_DMAMUX1_SYNC_LPTIM2_OUT	Synchronization Signal is LPTIM2 OUT
HAL_DMAMUX1_SYNC_DSI_TE	Synchronization Signal is DSI Tearing Effect
HAL_DMAMUX1_SYNC_DSI_EOT	Synchronization Signal is DSI End of refresh
HAL_DMAMUX1_SYNC_DMA2D_EOT	Synchronization Signal is DMA2D End of Transfer
HAL_DMAMUX1_SYNC_LDTC_IT	Synchronization Signal is LDTC IT

22 HAL DSI Generic Driver

22.1 DSI Firmware driver registers structures

22.1.1 DSI_InitTypeDef

Data Fields

- ***uint32_t AutomaticClockLaneControl***
- ***uint32_t TXEscapeCkdiv***
- ***uint32_t NumberOfLanes***

Field Documentation

- ***uint32_t DSI_InitTypeDef::AutomaticClockLaneControl***
  Automatic clock lane control This parameter can be any value of ***DSI_Automatic_Clk_Lane_Control***
- ***uint32_t DSI_InitTypeDef::TXEscapeCkdiv***
  TX Escape clock division The values 0 and 1 stop the TX_ESC clock generation
- ***uint32_t DSI_InitTypeDef::NumberOfLanes***
  Number of lanes This parameter can be any value of ***DSI_Number_Of_Lanes***

22.1.2 DSI_PLLInitTypeDef

Data Fields

- ***uint32_t PLLNDIV***
- ***uint32_t PLLIDF***
- ***uint32_t PLLODF***

Field Documentation

- ***uint32_t DSI_PLLInitTypeDef::PLLNDIV***
  PLL Loop Division Factor This parameter must be a value between 10 and 125
- ***uint32_t DSI_PLLInitTypeDef::PLLIDF***
  PLL Input Division Factor This parameter can be any value of ***DSI_PLL_IDF***
- ***uint32_t DSI_PLLInitTypeDef::PLLODF***
  PLL Output Division Factor This parameter can be any value of ***DSI_PLL_ODF***

22.1.3 DSI_VidCfgTypeDef

Data Fields

- ***uint32_t VirtualChannelID***
- ***uint32_t ColorCoding***
- ***uint32_t LooselyPacked***
- ***uint32_t Mode***
- ***uint32_t PacketSize***
- ***uint32_t NumberOfChunks***
- ***uint32_t NullPacketSize***
- ***uint32_t HSPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t DEPolarity***
- ***uint32_t HorizontalSyncActive***
- ***uint32_t HorizontalBackPorch***

- ***uint32_t HorizontalLine***
- ***uint32_t VerticalSyncActive***
- ***uint32_t VerticalBackPorch***
- ***uint32_t VerticalFrontPorch***
- ***uint32_t VerticalActive***
- ***uint32_t LPCommandEnable***
- ***uint32_t LPLargestPacketSize***
- ***uint32_t LPVACTLargestPacketSize***
- ***uint32_t LPHorizontalFrontPorchEnable***
- ***uint32_t LPHorizontalBackPorchEnable***
- ***uint32_t LPVerticalActiveEnable***
- ***uint32_t LPVerticalFrontPorchEnable***
- ***uint32_t LPVerticalBackPorchEnable***
- ***uint32_t LPVerticalSyncActiveEnable***
- ***uint32_t FrameBTAAcknowledgeEnable***

Field Documentation

- ***uint32_t DSI_VidCfgTypeDef::VirtualChannelID***
  Virtual channel ID
- ***uint32_t DSI_VidCfgTypeDef::ColorCoding***
  Color coding for LTDC interface This parameter can be any value of ***DSI_Color_Coding***
- ***uint32_t DSI_VidCfgTypeDef::LooselyPacked***
  Enable or disable loosely packed stream (needed only when using 18-bit configuration). This parameter can be any value of ***DSI_LooselyPacked***
- ***uint32_t DSI_VidCfgTypeDef::Mode***
  Video mode type This parameter can be any value of ***DSI_Video_Mode_Type***
- ***uint32_t DSI_VidCfgTypeDef::PacketSize***
  Video packet size
- ***uint32_t DSI_VidCfgTypeDef::NumberOfChunks***
  Number of chunks
- ***uint32_t DSI_VidCfgTypeDef::NullPacketSize***
  Null packet size
- ***uint32_t DSI_VidCfgTypeDef::HSPolarity***
  HSYNC pin polarity This parameter can be any value of ***DSI_HSYNC_Polarity***
- ***uint32_t DSI_VidCfgTypeDef::VSPolarity***
  VSYNC pin polarity This parameter can be any value of ***DSI_VSYNC_Polarity***
- ***uint32_t DSI_VidCfgTypeDef::DEPolarity***
  Data Enable pin polarity This parameter can be any value of ***DSI_DATA_ENABLE_Polarity***
- ***uint32_t DSI_VidCfgTypeDef::HorizontalSyncActive***
  Horizontal synchronism active duration (in lane byte clock cycles)
- ***uint32_t DSI_VidCfgTypeDef::HorizontalBackPorch***
  Horizontal back-porch duration (in lane byte clock cycles)
- ***uint32_t DSI_VidCfgTypeDef::HorizontalLine***
  Horizontal line duration (in lane byte clock cycles)
- ***uint32_t DSI_VidCfgTypeDef::VerticalSyncActive***
  Vertical synchronism active duration
- ***uint32_t DSI_VidCfgTypeDef::VerticalBackPorch***
  Vertical back-porch duration
- ***uint32_t DSI_VidCfgTypeDef::VerticalFrontPorch***
  Vertical front-porch duration
- ***uint32_t DSI_VidCfgTypeDef::VerticalActive***
  Vertical active duration

- ***uint32_t DSI_VidCfgTypeDef::LPCommandEnable***
  Low-power command enable This parameter can be any value of ***DSI_LP_Command***
- ***uint32_t DSI_VidCfgTypeDef::LPLargestPacketSize***
  The size, in bytes, of the low power largest packet that can fit in a line during VSA, VBP and VFP regions
- ***uint32_t DSI_VidCfgTypeDef::LPVACTLargestPacketSize***
  The size, in bytes, of the low power largest packet that can fit in a line during VACT region
- ***uint32_t DSI_VidCfgTypeDef::LPHorizontalFrontPorchEnable***
  Low-power horizontal front-porch enable This parameter can be any value of ***DSI_LP_HFP***
- ***uint32_t DSI_VidCfgTypeDef::LPHorizontalBackPorchEnable***
  Low-power horizontal back-porch enable This parameter can be any value of ***DSI_LP_HBP***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalActiveEnable***
  Low-power vertical active enable This parameter can be any value of ***DSI_LP_VACT***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalFrontPorchEnable***
  Low-power vertical front-porch enable This parameter can be any value of ***DSI_LP_VFP***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalBackPorchEnable***
  Low-power vertical back-porch enable This parameter can be any value of ***DSI_LP_VBP***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalSyncActiveEnable***
  Low-power vertical sync active enable This parameter can be any value of ***DSI_LP_VSYNC***
- ***uint32_t DSI_VidCfgTypeDef::FrameBTAAcknowledgeEnable***
  Frame bus-turn-around acknowledge enable This parameter can be any value of ***DSI_FBTA_acknowledge***

22.1.4 DSI_CmdCfgTypeDef

Data Fields

- ***uint32_t VirtualChannelID***
- ***uint32_t ColorCoding***
- ***uint32_t CommandSize***
- ***uint32_t TearingEffectSource***
- ***uint32_t TearingEffectPolarity***
- ***uint32_t HSPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t DEPolarity***
- ***uint32_t VSyncPol***
- ***uint32_t AutomaticRefresh***
- ***uint32_t TEAcknowledgeRequest***

Field Documentation

- ***uint32_t DSI_CmdCfgTypeDef::VirtualChannelID***
  Virtual channel ID
- ***uint32_t DSI_CmdCfgTypeDef::ColorCoding***
  Color coding for LTDC interface This parameter can be any value of ***DSI_Color_Coding***
- ***uint32_t DSI_CmdCfgTypeDef::CommandSize***
  Maximum allowed size for an LTDC write memory command, measured in pixels. This parameter can be any value between 0x00 and 0xFFFF

- ***uint32_t DSI_CmdCfgTypeDef::TearingEffectSource***
  Tearing effect source This parameter can be any value of ***DSI_TearingEffectSource***
- ***uint32_t DSI_CmdCfgTypeDef::TearingEffectPolarity***
  Tearing effect pin polarity This parameter can be any value of ***DSI_TearingEffectPolarity***
- ***uint32_t DSI_CmdCfgTypeDef::HSPolarity***
  HSYNC pin polarity This parameter can be any value of ***DSI_HSYNC_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::VSPolarity***
  VSYNC pin polarity This parameter can be any value of ***DSI_VSYNC_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::DEPolarity***
  Data Enable pin polarity This parameter can be any value of ***DSI_DATA_ENABLE_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::VSyncPol***
  VSync edge on which the LTDC is halted This parameter can be any value of ***DSI_VSync_Edge_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::AutomaticRefresh***
  Automatic refresh mode This parameter can be any value of ***DSI_AutomaticRefresh***
- ***uint32_t DSI_CmdCfgTypeDef::TEAcknowledgeRequest***
  Tearing Effect Acknowledge Request Enable This parameter can be any value of ***DSI_TE_AcknowledgeRequest***

22.1.5 DSI_LPCmdTypeDef

Data Fields

- ***uint32_t LPGenShortWriteNoP***
- ***uint32_t LPGenShortWriteOneP***
- ***uint32_t LPGenShortWriteTwoP***
- ***uint32_t LPGenShortReadNoP***
- ***uint32_t LPGenShortReadOneP***
- ***uint32_t LPGenShortReadTwoP***
- ***uint32_t LPGenLongWrite***
- ***uint32_t LPDcsShortWriteNoP***
- ***uint32_t LPDcsShortWriteOneP***
- ***uint32_t LPDcsShortReadNoP***
- ***uint32_t LPDcsLongWrite***
- ***uint32_t LPMaxReadPacket***
- ***uint32_t AcknowledgeRequest***

Field Documentation

- ***uint32_t DSI_LPCmdTypeDef::LPGenShortWriteNoP***
  Generic Short Write Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortWriteNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortWriteOneP***
  Generic Short Write One parameter Transmission This parameter can be any value of ***DSI_LP_LPGenShortWriteOneP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortWriteTwoP***
  Generic Short Write Two parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortWriteTwoP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortReadNoP***
  Generic Short Read Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortReadNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortReadOneP***
  Generic Short Read One parameter Transmission This parameter can be any value of ***DSI_LP_LPGenShortReadOneP***

- ***uint32_t DSI_LPCmdTypeDef::LPGenShortReadTwoP***
  Generic Short Read Two parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortReadTwoP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenLongWrite***
  Generic Long Write Transmission This parameter can be any value of ***DSI_LP_LPGenLongWrite***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsShortWriteNoP***
  DCS Short Write Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPDcsShortWriteNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsShortWriteOneP***
  DCS Short Write One parameter Transmission This parameter can be any value of ***DSI_LP_LPDcsShortWriteOneP***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsShortReadNoP***
  DCS Short Read Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPDcsShortReadNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsLongWrite***
  DCS Long Write Transmission This parameter can be any value of ***DSI_LP_LPDcsLongWrite***
- ***uint32_t DSI_LPCmdTypeDef::LPMaxReadPacket***
  Maximum Read Packet Size Transmission This parameter can be any value of ***DSI_LP_LPMaxReadPacket***
- ***uint32_t DSI_LPCmdTypeDef::AcknowledgeRequest***
  Acknowledge Request Enable This parameter can be any value of ***DSI_AcknowledgeRequest***

22.1.6 DSI_PHY_TimerTypeDef

Data Fields

- ***uint32_t ClockLaneHS2LPTime***
- ***uint32_t ClockLaneLP2HSTime***
- ***uint32_t DataLaneHS2LPTime***
- ***uint32_t DataLaneLP2HSTime***
- ***uint32_t DataLaneMaxReadTime***
- ***uint32_t StopWaitTime***

Field Documentation

- ***uint32_t DSI_PHY_TimerTypeDef::ClockLaneHS2LPTime***
  The maximum time that the D-PHY clock lane takes to go from high-speed to low-power transmission
- ***uint32_t DSI_PHY_TimerTypeDef::ClockLaneLP2HSTime***
  The maximum time that the D-PHY clock lane takes to go from low-power to high-speed transmission
- ***uint32_t DSI_PHY_TimerTypeDef::DataLaneHS2LPTime***
  The maximum time that the D-PHY data lanes takes to go from high-speed to low-power transmission
- ***uint32_t DSI_PHY_TimerTypeDef::DataLaneLP2HSTime***
  The maximum time that the D-PHY data lanes takes to go from low-power to high-speed transmission
- ***uint32_t DSI_PHY_TimerTypeDef::DataLaneMaxReadTime***
  The maximum time required to perform a read command
- ***uint32_t DSI_PHY_TimerTypeDef::StopWaitTime***
  The minimum wait period to request a High-Speed transmission after the Stop state

22.1.7 DSI_HOST_TimeoutTypeDef

Data Fields

- ***uint32_t TimeoutCkdiv***
- ***uint32_t HighSpeedTransmissionTimeout***
- ***uint32_t LowPowerReceptionTimeout***
- ***uint32_t HighSpeedReadTimeout***
- ***uint32_t LowPowerReadTimeout***
- ***uint32_t HighSpeedWriteTimeout***
- ***uint32_t HighSpeedWritePrespMode***
- ***uint32_t LowPowerWriteTimeout***
- ***uint32_t BTATimeout***

Field Documentation

- ***uint32_t DSI_HOST_TimeoutTypeDef::TimeoutCkdiv***
  Time-out clock division
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedTransmissionTimeout***
  High-speed transmission time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::LowPowerReceptionTimeout***
  Low-power reception time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedReadTimeout***
  High-speed read time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::LowPowerReadTimeout***
  Low-power read time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedWriteTimeout***
  High-speed write time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedWritePrespMode***
  High-speed write presp mode This parameter can be any value of ***DSI_HS_PrespMode***
- ***uint32_t DSI_HOST_TimeoutTypeDef::LowPowerWriteTimeout***
  Low-speed write time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::BTATimeout***
  BTA time-out

22.1.8 DSI_HandleTypeDef

Data Fields

- ***DSI_TypeDef * Instance***
- ***DSI_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DSI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t ErrorMsk***

Field Documentation

- ***DSI_TypeDef* DSI_HandleTypeDef::Instance***
  Register base address
- ***DSI_InitTypeDef DSI_HandleTypeDef::Init***
  DSI required parameters
- ***HAL_LockTypeDef DSI_HandleTypeDef::Lock***
  DSI peripheral status
- ***__IO HAL_DSI_StateTypeDef DSI_HandleTypeDef::State***
  DSI communication state

- ***__IO uint32_t DSI_HandleTypeDef::ErrorCode***
  DSI Error code
- ***uint32_t DSI_HandleTypeDef::ErrorMsk***
  DSI Error monitoring mask

22.2 DSI Firmware driver API description

22.2.1 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DSI
- De-initialize the DSI

This section contains the following APIs:

- ***HAL_DSI_Init()***
- ***HAL_DSI_DeInit()***
- ***HAL_DSI_ConfigErrorMonitor()***
- ***HAL_DSI_MspInit()***
- ***HAL_DSI_MspDeInit()***

22.2.2 IO operation functions

This section provides function allowing to:

- Handle DSI interrupt request

This section contains the following APIs:

- ***HAL_DSI_IRQHandler()***
- ***HAL_DSI_TearingEffectCallback()***
- ***HAL_DSI_EndOfRefreshCallback()***
- ***HAL_DSI_ErrorCallback()***

22.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure the Generic interface read-back Virtual Channel ID
- Select video mode and configure the corresponding parameters
- Configure command transmission mode: High-speed or Low-power
- Configure the flow control
- Configure the DSI PHY timer
- Configure the DSI HOST timeout
- Configure the DSI HOST timeout
- Start/Stop the DSI module
- Refresh the display in command mode
- Controls the display color mode in Video mode
- Control the display shutdown in Video mode
- Write short DCS or short Generic command
- Write long DCS or long Generic command
- Read command (DCS or generic)
- Enter/Exit the Ultra Low Power Mode on data only (D-PHY PLL running)
- Enter/Exit the Ultra Low Power Mode on data only and clock (D-PHY PLL turned off)
- Start/Stop test pattern generation
- Slew-Rate And Delay Tuning
- Low-Power Reception Filter Tuning

- Activate an additional current path on all lanes to meet the SDDTx parameter
- Custom lane pins configuration
- Set custom timing for the PHY
- Force the Clock/Data Lane in TX Stop Mode
- Force LP Receiver in Low-Power Mode
- Force Data Lanes in RX Mode after a BTA
- Enable a pull-down on the lanes to prevent from floating states when unused
- Switch off the contention detection on data lanes

This section contains the following APIs:

- ***HAL_DSI_SetGenericVCID()***
- ***HAL_DSI_ConfigVideoMode()***
- ***HAL_DSI_ConfigAdaptedCommandMode()***
- ***HAL_DSI_ConfigCommand()***
- ***HAL_DSI_ConfigFlowControl()***
- ***HAL_DSI_ConfigPhyTimer()***
- ***HAL_DSI_ConfigHostTimeouts()***
- ***HAL_DSI_Start()***
- ***HAL_DSI_Stop()***
- ***HAL_DSI_Refresh()***
- ***HAL_DSI_ColorMode()***
- ***HAL_DSI_Shutdown()***
- ***HAL_DSI_ShortWrite()***
- ***HAL_DSI_LongWrite()***
- ***HAL_DSI_Read()***
- ***HAL_DSI_EnterULPMData()***
- ***HAL_DSI_ExitULPMData()***
- ***HAL_DSI_EnterULPM()***
- ***HAL_DSI_ExitULPM()***
- ***HAL_DSI_PatternGeneratorStart()***
- ***HAL_DSI_PatternGeneratorStop()***
- ***HAL_DSI_SetSlewRateAndDelayTuning()***
- ***HAL_DSI_SetLowPowerRXFilter()***
- ***HAL_DSI_SetSDD()***
- ***HAL_DSI_SetLanePinsConfiguration()***
- ***HAL_DSI_SetPHYTimings()***
- ***HAL_DSI_ForceTXStopMode()***
- ***HAL_DSI_ForceRXLowPower()***
- ***HAL_DSI_ForceDataLanesInRX()***
- ***HAL_DSI_SetPullDown()***
- ***HAL_DSI_SetContentionDetectionOff()***

22.2.4 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DSI state.
- Get error code.

This section contains the following APIs:

- ***HAL_DSI_GetState()***

- *HAL_DSI_GetError()*

22.2.5 Detailed description of functions

HAL_DSI_Init

Function name	**HAL_StatusTypeDef HAL_DSI_Init (DSI_HandleTypeDef * hdsi, DSI_PLLInitTypeDef * PLLInit)**
Function description	Initializes the DSI according to the specified parameters in the DSI_InitTypeDef and create the associated handle.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **PLLInit:** pointer to a DSI_PLLInitTypeDef structure that contains the PLL Clock structure definition for the DSI.
Return values	• **HAL:** status

HAL_DSI_DeInit

Function name	**HAL_StatusTypeDef HAL_DSI_DeInit (DSI_HandleTypeDef * hdsi)**
Function description	De-initializes the DSI peripheral registers to their default reset values.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **HAL:** status

HAL_DSI_MspInit

Function name	**void HAL_DSI_MspInit (DSI_HandleTypeDef * hdsi)**
Function description	Initializes the DSI MSP.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **None:**

HAL_DSI_MspDeInit

Function name	**void HAL_DSI_MspDeInit (DSI_HandleTypeDef * hdsi)**
Function description	De-initializes the DSI MSP.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **None:**

HAL_DSI_IRQHandler

Function name	**void HAL_DSI_IRQHandler (DSI_HandleTypeDef * hdsi)**
Function description	Handles DSI interrupt request.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that

contains the configuration information for the DSI.

Return values	• **HAL:** status

HAL_DSI_TearingEffectCallback

Function name	**void HAL_DSI_TearingEffectCallback (DSI_HandleTypeDef * hdsi)**
Function description	Tearing Effect DSI callback.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **None:**

HAL_DSI_EndOfRefreshCallback

Function name	**void HAL_DSI_EndOfRefreshCallback (DSI_HandleTypeDef * hdsi)**
Function description	End of Refresh DSI callback.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **None:**

HAL_DSI_ErrorCallback

Function name	**void HAL_DSI_ErrorCallback (DSI_HandleTypeDef * hdsi)**
Function description	Operation Error DSI callback.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **None:**

HAL_DSI_SetGenericVCID

Function name	**HAL_StatusTypeDef HAL_DSI_SetGenericVCID (DSI_HandleTypeDef * hdsi, uint32_t VirtualChannelID)**
Function description	Configure the Generic interface read-back Virtual Channel ID.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **VirtualChannelID:** Virtual channel ID
Return values	• **HAL:** status

HAL_DSI_ConfigVideoMode

Function name	**HAL_StatusTypeDef HAL_DSI_ConfigVideoMode (DSI_HandleTypeDef * hdsi, DSI_VidCfgTypeDef * VidCfg)**
Function description	Select video mode and configure the corresponding parameters.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

	• **VidCfg:** pointer to a DSI_VidCfgTypeDef structure that contains the DSI video mode configuration parameters
Return values	• **HAL:** status

HAL_DSI_ConfigAdaptedCommandMode

Function name	**HAL_StatusTypeDef HAL_DSI_ConfigAdaptedCommandMode (DSI_HandleTypeDef * hdsi, DSI_CmdCfgTypeDef * CmdCfg)**
Function description	Select adapted command mode and configure the corresponding parameters.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **CmdCfg:** pointer to a DSI_CmdCfgTypeDef structure that contains the DSI command mode configuration parameters
Return values	• **HAL:** status

HAL_DSI_ConfigCommand

Function name	**HAL_StatusTypeDef HAL_DSI_ConfigCommand (DSI_HandleTypeDef * hdsi, DSI_LPCmdTypeDef * LPCmd)**
Function description	Configure command transmission mode: High-speed or Low-power and enable/disable acknowledge request after packet transmission.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **LPCmd:** pointer to a DSI_LPCmdTypeDef structure that contains the DSI command transmission mode configuration parameters
Return values	• **HAL:** status

HAL_DSI_ConfigFlowControl

Function name	**HAL_StatusTypeDef HAL_DSI_ConfigFlowControl (DSI_HandleTypeDef * hdsi, uint32_t FlowControl)**
Function description	Configure the flow control parameters.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **FlowControl:** flow control feature(s) to be enabled. This parameter can be any combination of DSI Flow Control.
Return values	• **HAL:** status

HAL_DSI_ConfigPhyTimer

Function name	**HAL_StatusTypeDef HAL_DSI_ConfigPhyTimer (DSI_HandleTypeDef * hdsi, DSI_PHY_TimerTypeDef * PhyTimers)**
Function description	Configure the DSI PHY timer parameters.

Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **PhyTimers:** DSI_PHY_TimerTypeDef structure that contains the DSI PHY timing parameters
Return values	• **HAL:** status

HAL_DSI_ConfigHostTimeouts

Function name	**HAL_StatusTypeDef HAL_DSI_ConfigHostTimeouts (DSI_HandleTypeDef * hdsi, DSI_HOST_TimeoutTypeDef * HostTimeouts)**
Function description	Configure the DSI HOST timeout parameters.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **HostTimeouts:** DSI_HOST_TimeoutTypeDef structure that contains the DSI host timeout parameters
Return values	• **HAL:** status

HAL_DSI_Start

Function name	**HAL_StatusTypeDef HAL_DSI_Start (DSI_HandleTypeDef * hdsi)**
Function description	Start the DSI module.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **HAL:** status

HAL_DSI_Stop

Function name	**HAL_StatusTypeDef HAL_DSI_Stop (DSI_HandleTypeDef * hdsi)**
Function description	Stop the DSI module.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **HAL:** status

HAL_DSI_Refresh

Function name	**HAL_StatusTypeDef HAL_DSI_Refresh (DSI_HandleTypeDef * hdsi)**
Function description	Refresh the display in command mode.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **HAL:** status

HAL_DSI_ColorMode

Function name	**HAL_StatusTypeDef HAL_DSI_ColorMode (DSI_HandleTypeDef * hdsi, uint32_t ColorMode)**
Function description	Controls the display color mode in Video mode.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **ColorMode:** Color mode (full or 8-colors). This parameter can be any value of DSI Color Mode
Return values	• **HAL:** status

HAL_DSI_Shutdown

Function name	**HAL_StatusTypeDef HAL_DSI_Shutdown (DSI_HandleTypeDef * hdsi, uint32_t Shutdown)**
Function description	Control the display shutdown in Video mode.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **Shutdown:** Shut-down (Display-ON or Display-OFF). This parameter can be any value of DSI ShutDown
Return values	• **HAL:** status

HAL_DSI_ShortWrite

Function name	**HAL_StatusTypeDef HAL_DSI_ShortWrite (DSI_HandleTypeDef * hdsi, uint32_t ChannelID, uint32_t Mode, uint32_t Param1, uint32_t Param2)**
Function description	DCS or Generic short write command.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **ChannelID:** Virtual channel ID. • **Mode:** DSI short packet data type. This parameter can be any value of DSI SHORT WRITE PKT Data Type. • **Param1:** DSC command or first generic parameter. This parameter can be any value of DSI DCS Command or a generic command code. • **Param2:** DSC parameter or second generic parameter.
Return values	• **HAL:** status

HAL_DSI_LongWrite

Function name	**HAL_StatusTypeDef HAL_DSI_LongWrite (DSI_HandleTypeDef * hdsi, uint32_t ChannelID, uint32_t Mode, uint32_t NbParams, uint32_t Param1, uint8_t * ParametersTable)**
Function description	DCS or Generic long write command.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **ChannelID:** Virtual channel ID.

- **Mode:** DSI long packet data type. This parameter can be any value of DSI LONG WRITE PKT Data Type.
- **NbParams:** Number of parameters.
- **Param1:** DSC command or first generic parameter. This parameter can be any value of DSI DCS Command or a generic command code
- **ParametersTable:** Pointer to parameter values table.

Return values	• **HAL:** status

HAL_DSI_Read

Function name	**HAL_StatusTypeDef HAL_DSI_Read (DSI_HandleTypeDef * hdsi, uint32_t ChannelNbr, uint8_t * Array, uint32_t Size, uint32_t Mode, uint32_t DCSCmd, uint8_t * ParametersTable)**
Function description	Read command (DCS or generic)
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **ChannelNbr:** Virtual channel ID • **Array:** pointer to a buffer to store the payload of a read back operation. • **Size:** Data size to be read (in byte). • **Mode:** DSI read packet data type. This parameter can be any value of DSI SHORT READ PKT Data Type. • **DCSCmd:** DCS get/read command. • **ParametersTable:** Pointer to parameter values table.
Return values	• **HAL:** status

HAL_DSI_EnterULPMData

Function name	**HAL_StatusTypeDef HAL_DSI_EnterULPMData (DSI_HandleTypeDef * hdsi)**
Function description	Enter the ULPM (Ultra Low Power Mode) with the D-PHY PLL running (only data lanes are in ULPM)
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **HAL:** status

HAL_DSI_ExitULPMData

Function name	**HAL_StatusTypeDef HAL_DSI_ExitULPMData (DSI_HandleTypeDef * hdsi)**
Function description	Exit the ULPM (Ultra Low Power Mode) with the D-PHY PLL running (only data lanes are in ULPM)
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **HAL:** status

HAL_DSI_EnterULPM

Function name	**HAL_StatusTypeDef HAL_DSI_EnterULPM (DSI_HandleTypeDef * hdsi)**
Function description	Enter the ULPM (Ultra Low Power Mode) with the D-PHY PLL turned off (both data and clock lanes are in ULPM)
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **HAL:** status

HAL_DSI_ExitULPM

Function name	**HAL_StatusTypeDef HAL_DSI_ExitULPM (DSI_HandleTypeDef * hdsi)**
Function description	Exit the ULPM (Ultra Low Power Mode) with the D-PHY PLL turned off (both data and clock lanes are in ULPM)
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **HAL:** status

HAL_DSI_PatternGeneratorStart

Function name	**HAL_StatusTypeDef HAL_DSI_PatternGeneratorStart (DSI_HandleTypeDef * hdsi, uint32_t Mode, uint32_t Orientation)**
Function description	Start test pattern generation.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **Mode:** Pattern generator mode This parameter can be one of the following values: 0: Color bars (horizontal or vertical) 1: BER pattern (vertical only) • **Orientation:** Pattern generator orientation This parameter can be one of the following values: 0: Vertical color bars 1: Horizontal color bars
Return values	• **HAL:** status

HAL_DSI_PatternGeneratorStop

Function name	**HAL_StatusTypeDef HAL_DSI_PatternGeneratorStop (DSI_HandleTypeDef * hdsi)**
Function description	Stop test pattern generation.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **HAL:** status

HAL_DSI_SetSlewRateAndDelayTuning

Function name	**HAL_StatusTypeDef HAL_DSI_SetSlewRateAndDelayTuning**

	(DSI_HandleTypeDef * hdsi, uint32_t CommDelay, uint32_t Lane, uint32_t Value)
Function description	Set Slew-Rate And Delay Tuning.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **CommDelay:** Communication delay to be adjusted. This parameter can be any value of DSI Communication Delay • **Lane:** select between clock or data lanes. This parameter can be any value of DSI Lane Group • **Value:** Custom value of the slew-rate or delay
Return values	• **HAL:** status

HAL_DSI_SetLowPowerRXFilter

Function name	**HAL_StatusTypeDef HAL_DSI_SetLowPowerRXFilter (DSI_HandleTypeDef * hdsi, uint32_t Frequency)**
Function description	Low-Power Reception Filter Tuning.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **Frequency:** cutoff frequency of low-pass filter at the input of LPRX
Return values	• **HAL:** status

HAL_DSI_SetSDD

Function name	**HAL_StatusTypeDef HAL_DSI_SetSDD (DSI_HandleTypeDef * hdsi, FunctionalState State)**
Function description	Activate an additional current path on all lanes to meet the SDDTx parameter defined in the MIPI D-PHY specification.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **State:** ENABLE or DISABLE
Return values	• **HAL:** status

HAL_DSI_SetLanePinsConfiguration

Function name	**HAL_StatusTypeDef HAL_DSI_SetLanePinsConfiguration (DSI_HandleTypeDef * hdsi, uint32_t CustomLane, uint32_t Lane, FunctionalState State)**
Function description	Custom lane pins configuration.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **CustomLane:** Function to be applyed on selected lane. This parameter can be any value of DSI CustomLane • **Lane:** select between clock or data lane 0 or data lane 1. This parameter can be any value of DSI Lane Select • **State:** ENABLE or DISABLE

Return values	• **HAL:** status

HAL_DSI_SetPHYTimings

Function name	**HAL_StatusTypeDef HAL_DSI_SetPHYTimings (DSI_HandleTypeDef * hdsi, uint32_t Timing, FunctionalState State, uint32_t Value)**
Function description	Set custom timing for the PHY.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **Timing:** PHY timing to be adjusted. This parameter can be any value of DSI PHY Timing • **State:** ENABLE or DISABLE • **Value:** Custom value of the timing
Return values	• **HAL:** status

HAL_DSI_ForceTXStopMode

Function name	**HAL_StatusTypeDef HAL_DSI_ForceTXStopMode (DSI_HandleTypeDef * hdsi, uint32_t Lane, FunctionalState State)**
Function description	Force the Clock/Data Lane in TX Stop Mode.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **Lane:** select between clock or data lanes. This parameter can be any value of DSI Lane Group • **State:** ENABLE or DISABLE
Return values	• **HAL:** status

HAL_DSI_ForceRXLowPower

Function name	**HAL_StatusTypeDef HAL_DSI_ForceRXLowPower (DSI_HandleTypeDef * hdsi, FunctionalState State)**
Function description	Forces LP Receiver in Low-Power Mode.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **State:** ENABLE or DISABLE
Return values	• **HAL:** status

HAL_DSI_ForceDataLanesInRX

Function name	**HAL_StatusTypeDef HAL_DSI_ForceDataLanesInRX (DSI_HandleTypeDef * hdsi, FunctionalState State)**
Function description	Force Data Lanes in RX Mode after a BTA.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **State:** ENABLE or DISABLE

Return values	• **HAL:** status

HAL_DSI_SetPullDown

Function name	**HAL_StatusTypeDef HAL_DSI_SetPullDown (DSI_HandleTypeDef * hdsi, FunctionalState State)**
Function description	Enable a pull-down on the lanes to prevent from floating states when unused.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **State:** ENABLE or DISABLE
Return values	• **HAL:** status

HAL_DSI_SetContentionDetectionOff

Function name	**HAL_StatusTypeDef HAL_DSI_SetContentionDetectionOff (DSI_HandleTypeDef * hdsi, FunctionalState State)**
Function description	Switch off the contention detection on data lanes.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **State:** ENABLE or DISABLE
Return values	• **HAL:** status

HAL_DSI_GetError

Function name	**uint32_t HAL_DSI_GetError (DSI_HandleTypeDef * hdsi)**
Function description	Return the DSI error code.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **DSI:** Error Code

HAL_DSI_ConfigErrorMonitor

Function name	**HAL_StatusTypeDef HAL_DSI_ConfigErrorMonitor (DSI_HandleTypeDef * hdsi, uint32_t ActiveErrors)**
Function description	Enable the error monitor flags.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI. • **ActiveErrors:** indicates which error interrupts will be enabled. This parameter can be any combination of DSI Error Data Type.
Return values	• **HAL:** status

HAL_DSI_GetState

Function name	**HAL_DSI_StateTypeDef HAL_DSI_GetState (DSI_HandleTypeDef * hdsi)**

Function description	Return the DSI state.
Parameters	• **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
Return values	• **HAL:** state

22.3 DSI Firmware driver defines

22.3.1 DSI

DSI Acknowledge Request

DSI_ACKNOWLEDGE_DISABLE

DSI_ACKNOWLEDGE_ENABLE

DSI Automatic Refresh

DSI_AR_DISABLE

DSI_AR_ENABLE

DSI Automatic Clk Lane Control

DSI_AUTO_CLK_LANE_CTRL_DISABLE

DSI_AUTO_CLK_LANE_CTRL_ENABLE

DSI Color Coding

DSI_RGB565	The values 0x00000001 and 0x00000002 can also be used for the RGB565 color mode configuration
DSI_RGB666	The value 0x00000004 can also be used for the RGB666 color mode configuration
DSI_RGB888	

DSI Color Mode

DSI_COLOR_MODE_FULL

DSI_COLOR_MODE_EIGHT

DSI Communication Delay

DSI_SLEW_RATE_HSTX

DSI_SLEW_RATE_LPTX

DSI_HS_DELAY

DSI CustomLane

DSI_SWAP_LANE_PINS

DSI_INVERT_HS_SIGNAL

DSI DATA ENABLE Polarity

DSI_DATA_ENABLE_ACTIVE_HIGH

DSI_DATA_ENABLE_ACTIVE_LOW

DSI DCS Command

DSI_ENTER_IDLE_MODE

DSI_ENTER_INVERT_MODE

DSI_ENTER_NORMAL_MODE

DSI_ENTER_PARTIAL_MODE

DSI_ENTER_SLEEP_MODE

DSI_EXIT_IDLE_MODE

DSI_EXIT_INVERT_MODE

DSI_EXIT_SLEEP_MODE

DSI_GET_3D_CONTROL

DSI_GET_ADDRESS_MODE

DSI_GET_BLUE_CHANNEL

DSI_GET_DIAGNOSTIC_RESULT

DSI_GET_DISPLAY_MODE

DSI_GET_GREEN_CHANNEL

DSI_GET_PIXEL_FORMAT

DSI_GET_POWER_MODE

DSI_GET_RED_CHANNEL

DSI_GET_SCANLINE

DSI_GET_SIGNAL_MODE

DSI_NOP

DSI_READ_DDB_CONTINUE

DSI_READ_DDB_START

DSI_READ_MEMORY_CONTINUE

DSI_READ_MEMORY_START

DSI_SET_3D_CONTROL

DSI_SET_ADDRESS_MODE

DSI_SET_COLUMN_ADDRESS

DSI_SET_DISPLAY_OFF

DSI_SET_DISPLAY_ON

DSI_SET_GAMMA_CURVE

DSI_SET_PAGE_ADDRESS

DSI_SET_PARTIAL_COLUMNS

DSI_SET_PARTIAL_ROWS

DSI_SET_PIXEL_FORMAT

DSI_SET_SCROLL_AREA

DSI_SET_SCROLL_START

DSI_SET_TEAR_OFF

DSI_SET_TEAR_ON

DSI_SET_TEAR_SCANLINE

DSI_SET_VSYNC_TIMING

DSI_SOFT_RESET

DSI_WRITE_LUT

DSI_WRITE_MEMORY_CONTINUE

DSI_WRITE_MEMORY_START

DSI Error Data Type

HAL_DSI_ERROR_NONE	
HAL_DSI_ERROR_ACK	acknowledge errors
HAL_DSI_ERROR_PHY	PHY related errors
HAL_DSI_ERROR_TX	transmission error
HAL_DSI_ERROR_RX	reception error
HAL_DSI_ERROR_ECC	ECC errors
HAL_DSI_ERROR_CRC	CRC error
HAL_DSI_ERROR_PSE	Packet Size error
HAL_DSI_ERROR_EOT	End Of Transmission error
HAL_DSI_ERROR_OVF	FIFO overflow error
HAL_DSI_ERROR_GEN	Generic FIFO related errors

DSI FBTA Acknowledge

DSI_FBTAA_DISABLE

DSI_FBTAA_ENABLE

DSI Flags

DSI_FLAG_TE

DSI_FLAG_ER

DSI_FLAG_BUSY

DSI_FLAG_PLLLS

DSI_FLAG_PLLL

DSI_FLAG_PLLU

DSI_FLAG_RRS

DSI_FLAG_RR

DSI Flow Control

DSI_FLOW_CONTROL_CRC_RX

DSI_FLOW_CONTROL_ECC_RX

DSI_FLOW_CONTROL_BTA

DSI_FLOW_CONTROL_EOTP_RX

DSI_FLOW_CONTROL_EOTP_TX

DSI_FLOW_CONTROL_ALL

DSI HSYNC Polarity

DSI_HSYNC_ACTIVE_HIGH

DSI_HSYNC_ACTIVE_LOW

DSI HS Presp Mode

DSI_HS_PM_DISABLE

DSI_HS_PM_ENABLE

DSI Interrupts

DSI_IT_TE

DSI_IT_ER

DSI_IT_PLLL

DSI_IT_PLLU

DSI_IT_RR

DSI Lane Group

DSI_CLOCK_LANE

DSI_CLOCK_LANE

DSI_DATA_LANES

DSI Lane Select

DSI_DATA_LANE0

DSI_DATA_LANE1

DSI LONG WRITE PKT Data Type

DSI_DCS_LONG_PKT_WRITE	DCS long write
DSI_GEN_LONG_PKT_WRITE	Generic long write

DSI Loosely Packed

DSI_LOOSELY_PACKED_ENABLE

DSI_LOOSELY_PACKED_DISABLE

DSI LP Command

DSI_LP_COMMAND_DISABLE

DSI_LP_COMMAND_ENABLE

DSI LP HBP

DSI_LP_HBP_DISABLE

DSI_LP_HBP_ENABLE

DSI LP HFP

DSI_LP_HFP_DISABLE

DSI_LP_HFP_ENABLE

DSI LP LPDcs Long Write

DSI_LP_DLW_DISABLE

DSI_LP_DLW_ENABLE

DSI LP LPDcs Short Read NoP

DSI_LP_DSR0P_DISABLE

DSI_LP_DSR0P_ENABLE

DSI LP LPDcs Short Write NoP

DSI_LP_DSW0P_DISABLE

DSI_LP_DSW0P_ENABLE

DSI LP LPDcs Short Write OneP

DSI_LP_DSW1P_DISABLE

DSI_LP_DSW1P_ENABLE

DSI LP LPGen LongWrite

DSI_LP_GLW_DISABLE

DSI_LP_GLW_ENABLE

DSI LP LPGen Short Read NoP

DSI_LP_GSR0P_DISABLE

DSI_LP_GSR0P_ENABLE

DSI LP LPGen Short Read OneP

DSI_LP_GSR1P_DISABLE

DSI_LP_GSR1P_ENABLE

DSI LP LPGen Short Read TwoP

DSI_LP_GSR2P_DISABLE

DSI_LP_GSR2P_ENABLE

DSI LP LPGen Short Write NoP

DSI_LP_GSW0P_DISABLE

DSI_LP_GSW0P_ENABLE

DSI LP LPGen Short Write OneP

DSI_LP_GSW1P_DISABLE

DSI_LP_GSW1P_ENABLE

DSI LP LPGen Short Write TwoP

DSI_LP_GSW2P_DISABLE

DSI_LP_GSW2P_ENABLE

DSI LP LPMax Read Packet

DSI_LP_MRDP_DISABLE

DSI_LP_MRDP_ENABLE

DSI LP VACT

DSI_LP_VACT_DISABLE

DSI_LP_VACT_ENABLE

DSI LP VBP

DSI_LP_VBP_DISABLE

DSI_LP_VBP_ENABLE

DSI LP VFP

DSI_LP_VFP_DISABLE

DSI_LP_VFP_ENABLE

DSI LP VSYNC

DSI_LP_VSYNC_DISABLE

DSI_LP_VSYNC_ENABLE

DSI Number Of Lanes

DSI_ONE_DATA_LANE

DSI_TWO_DATA_LANES

DSI PHY Timing

DSI_TCLK_POST

DSI_TLPX_CLK

DSI_THS_EXIT

DSI_TLPX_DATA

DSI_THS_ZERO

DSI_THS_TRAIL

DSI_THS_PREPARE

DSI_TCLK_ZERO

DSI_TCLK_PREPARE

DSI PLL IDF

DSI_PLL_IN_DIV1

DSI_PLL_IN_DIV2

DSI_PLL_IN_DIV3

DSI_PLL_IN_DIV4

DSI_PLL_IN_DIV5

DSI_PLL_IN_DIV6

DSI_PLL_IN_DIV7

DSI PLL ODF

DSI_PLL_OUT_DIV1

DSI_PLL_OUT_DIV2

DSI_PLL_OUT_DIV4

DSI_PLL_OUT_DIV8

DSI SHORT READ PKT Data Type

DSI_DCS_SHORT_PKT_READ	DCS short read
DSI_GEN_SHORT_PKT_READ_P0	Generic short read, no parameters
DSI_GEN_SHORT_PKT_READ_P1	Generic short read, one parameter
DSI_GEN_SHORT_PKT_READ_P2	Generic short read, two parameters

DSI SHORT WRITE PKT Data Type

DSI_DCS_SHORT_PKT_WRITE_P0	DCS short write, no parameters
DSI_DCS_SHORT_PKT_WRITE_P1	DCS short write, one parameter
DSI_GEN_SHORT_PKT_WRITE_P0	Generic short write, no parameters
DSI_GEN_SHORT_PKT_WRITE_P1	Generic short write, one parameter
DSI_GEN_SHORT_PKT_WRITE_P2	Generic short write, two parameters

DSI ShutDown

DSI_DISPLAY_ON

DSI_DISPLAY_OFF

DSI Tearing Effect Polarity

DSI_TE_RISING_EDGE

DSI_TE_FALLING_EDGE

DSI Tearing Effect Source

DSI_TE_DSILINK

DSI_TE_EXTERNAL

DSI TE Acknowledge Request

DSI_TE_ACKNOWLEDGE_DISABLE

DSI_TE_ACKNOWLEDGE_ENABLE

DSI Video Mode Type

DSI_VID_MODE_NB_PULSES

DSI_VID_MODE_NB_EVENTS

DSI_VID_MODE_BURST

DSI VSync Edge Polarity

DSI_VSYNC_FALLING

DSI_VSYNC_RISING

DSI VSYNC Polarity

DSI_VSYNC_ACTIVE_HIGH

DSI_VSYNC_ACTIVE_LOW

23 HAL FIREWALL Generic Driver

23.1 FIREWALL Firmware driver registers structures

23.1.1 FIREWALL_InitTypeDef

Data Fields

- ***uint32_t CodeSegmentStartAddress***
- ***uint32_t CodeSegmentLength***
- ***uint32_t NonVDataSegmentStartAddress***
- ***uint32_t NonVDataSegmentLength***
- ***uint32_t VDataSegmentStartAddress***
- ***uint32_t VDataSegmentLength***
- ***uint32_t VolatileDataExecution***
- ***uint32_t VolatileDataShared***

Field Documentation

- ***uint32_t FIREWALL_InitTypeDef::CodeSegmentStartAddress***
  Protected code segment start address. This value is 24-bit long, the 8 LSB bits are reserved and forced to 0 in order to allow a 256-byte granularity.
- ***uint32_t FIREWALL_InitTypeDef::CodeSegmentLength***
  Protected code segment length in bytes. This value is 22-bit long, the 8 LSB bits are reserved and forced to 0 for the length to be a multiple of 256 bytes.
- ***uint32_t FIREWALL_InitTypeDef::NonVDataSegmentStartAddress***
  Protected non-volatile data segment start address. This value is 24-bit long, the 8 LSB bits are reserved and forced to 0 in order to allow a 256-byte granularity.
- ***uint32_t FIREWALL_InitTypeDef::NonVDataSegmentLength***
  Protected non-volatile data segment length in bytes. This value is 22-bit long, the 8 LSB bits are reserved and forced to 0 for the length to be a multiple of 256 bytes.
- ***uint32_t FIREWALL_InitTypeDef::VDataSegmentStartAddress***
  Protected volatile data segment start address. This value is 17-bit long, the 6 LSB bits are reserved and forced to 0 in order to allow a 64-byte granularity.
- ***uint32_t FIREWALL_InitTypeDef::VDataSegmentLength***
  Protected volatile data segment length in bytes. This value is 17-bit long, the 6 LSB bits are reserved and forced to 0 for the length to be a multiple of 64 bytes.
- ***uint32_t FIREWALL_InitTypeDef::VolatileDataExecution***
  Set VDE bit specifying whether or not the volatile data segment can be executed. When VDS = 1 (set by parameter VolatileDataShared), VDE bit has no meaning. This parameter can be a value of ***FIREWALL_VolatileData_Executable***
- ***uint32_t FIREWALL_InitTypeDef::VolatileDataShared***
  Set VDS bit in specifying whether or not the volatile data segment can be shared with a non-protected application code. This parameter can be a value of ***FIREWALL_VolatileData_Shared***

23.2 FIREWALL Firmware driver API description

23.2.1 How to use this driver

The FIREWALL HAL driver can be used as follows:

1. Declare a FIREWALL_InitTypeDef initialization structure.
2. Resort to HAL_FIREWALL_Config() API to initialize the Firewall

3. Enable the FIREWALL in calling HAL_FIREWALL_EnableFirewall() API
4. To ensure that any code executed outside the protected segment closes the FIREWALL, the user must set the flag FIREWALL_PRE_ARM_SET in calling __HAL_FIREWALL_PREARM_ENABLE() macro if called within a protected code segment or HAL_FIREWALL_EnablePreArmFlag() API if called outside of protected code segment after HAL_FIREWALL_Config() call.

23.2.2 Initialization and Configuration functions

This subsection provides the functions allowing to initialize the Firewall. Initialization is done by HAL_FIREWALL_Config():

- Enable the Firewall clock thru __HAL_RCC_FIREWALL_CLK_ENABLE() macro.
- Set the protected code segment address start and length.
- Set the protected non-volatile and/or volatile data segments address starts and lengths if applicable.
- Set the volatile data segment execution and sharing status.
- Length must be set to 0 for an unprotected segment.

This section contains the following APIs:

- *HAL_FIREWALL_Config()*
- *HAL_FIREWALL_GetConfig()*
- *HAL_FIREWALL_EnableFirewall()*
- *HAL_FIREWALL_EnablePreArmFlag()*
- *HAL_FIREWALL_DisablePreArmFlag()*

23.2.3 Detailed description of functions

HAL_FIREWALL_Config

Function name	**HAL_StatusTypeDef HAL_FIREWALL_Config (FIREWALL_InitTypeDef * fw_init)**
Function description	Initialize the Firewall according to the FIREWALL_InitTypeDef structure parameters.
Parameters	• **fw_init:** Firewall initialization structure
Return values	• **HAL:** status
Notes	• The API returns HAL_ERROR if the Firewall is already enabled.

HAL_FIREWALL_GetConfig

Function name	**void HAL_FIREWALL_GetConfig (FIREWALL_InitTypeDef * fw_config)**
Function description	Retrieve the Firewall configuration.
Parameters	• **fw_config:** Firewall configuration, type is same as initialization structure
Return values	• **None:**
Notes	• This API can't be executed inside a code area protected by the Firewall when the Firewall is enabled • If NVDSL register is different from 0, that is, if the non volatile data segment is defined, this API can't be executed when the

Firewall is enabled.

- User should resort to __HAL_FIREWALL_GET_PREARM() macro to retrieve FPA bit status

HAL_FIREWALL_EnableFirewall

Function name	**void HAL_FIREWALL_EnableFirewall (void)**
Function description	Enable FIREWALL.
Return values	• **None:**
Notes	• Firewall is enabled in clearing FWDIS bit of SYSCFG CFGR1 register. Once enabled, the Firewall cannot be disabled by software. Only a system reset can set again FWDIS bit.

HAL_FIREWALL_EnablePreArmFlag

Function name	**void HAL_FIREWALL_EnablePreArmFlag (void)**
Function description	Enable FIREWALL pre arm.
Return values	• **None:**
Notes	• When FPA bit is set, any code executed outside the protected segment will close the Firewall. • This API provides the same service as __HAL_FIREWALL_PREARM_ENABLE() macro but can't be executed inside a code area protected by the Firewall. • When the Firewall is disabled, user can resort to HAL_FIREWALL_EnablePreArmFlag() API any time. • When the Firewall is enabled and NVDSL register is equal to 0 (that is, when the non volatile data segment is not defined), ** this API can be executed when the Firewall is closed ** when the Firewall is opened, user should resort to __HAL_FIREWALL_PREARM_ENABLE() macro instead • When the Firewall is enabled and NVDSL register is different from 0 (that is, when the non volatile data segment is defined) ** FW_CR register can be accessed only when the Firewall is opened: user should resort to __HAL_FIREWALL_PREARM_ENABLE() macro instead.

HAL_FIREWALL_DisablePreArmFlag

Function name	**void HAL_FIREWALL_DisablePreArmFlag (void)**
Function description	Disable FIREWALL pre arm.
Return values	• **None:**
Notes	• When FPA bit is reset, any code executed outside the protected segment when the Firewall is opened will generate a system reset. • This API provides the same service as __HAL_FIREWALL_PREARM_DISABLE() macro but can't be executed inside a code area protected by the Firewall. • When the Firewall is disabled, user can resort to HAL_FIREWALL_EnablePreArmFlag() API any time.

- When the Firewall is enabled and NVDSL register is equal to 0 (that is, when the non volatile data segment is not defined), ** this API can be executed when the Firewall is closed ** when the Firewall is opened, user should resort to __HAL_FIREWALL_PREARM_DISABLE() macro instead
- When the Firewall is enabled and NVDSL register is different from 0 (that is, when the non volatile data segment is defined) ** FW_CR register can be accessed only when the Firewall is opened: user should resort to __HAL_FIREWALL_PREARM_DISABLE() macro instead.

23.3 FIREWALL Firmware driver defines

23.3.1 FIREWALL

FIREWALL Exported Macros

__HAL_FIREWALL_IS_ENABLED

Description:

- Check whether the FIREWALL is enabled or not.

Return value:

- FIREWALL: enabling status (TRUE or FALSE).

__HAL_FIREWALL_PREARM_ENABLE

Notes:

- When FPA bit is set, any code executed outside the protected segment closes the Firewall, otherwise it generates a system reset. This macro provides the same service as HAL_FIREWALL_EnablePreArmFlag() API but can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_PREARM_DISABLE

Notes:

- When FPA bit is set, any code executed outside the protected segment closes the Firewall, otherwise, it generates a system reset. This macro provides the same service as HAL_FIREWALL_DisablePreArmFlag() API but can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_VOLATILEDATA_SHARED_ENABLE	**Notes:** • When VDS bit is set, the volatile data segment is shared with non-protected application code. It can be accessed whatever the Firewall state (opened or closed). This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.
__HAL_FIREWALL_VOLATILEDATA_SHARED_DISABLE	**Notes:** • When VDS bit is reset, the volatile data segment is not shared and cannot be hit by a non protected executable code when the Firewall is closed. If it is accessed in such a condition, a system reset is generated by the Firewall. This macro can be executed inside a code area protected by the Firewall. This macro can be executed

whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_VOLATILEDATA_EXECUTION_ENABLE	**Notes:** • VDE bit is ignored when VDS is set. IF VDS = 1, the Volatile data segment can be executed whatever the VDE bit value. When VDE bit is set (with VDS = 0), the volatile data segment is executable. When the Firewall call is closed, a "call gate" entry procedure is required to open first the Firewall. This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.
__HAL_FIREWALL_VOLATILEDATA_EXECUTION_DISABLE	**Notes:** • VDE bit is ignored when VDS is set. IF VDS = 1, the Volatile data segment can be executed whatever the VDE bit value. When VDE bit is reset (with VDS = 0), the volatile data segment cannot be executed. This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0.

Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_GET_VOLATILEDATA_SHARED

Description:

- Check whether or not the volatile data segment is shared.

Return value:

- VDS: bit setting status (TRUE or FALSE).

Notes:

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_GET_VOLATILEDATA_EXECUTION

Description:

- Check whether or not the volatile data segment is declared executable.

Return value:

- VDE: bit setting status (TRUE or FALSE).

Notes:

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_GET_PREARM

Description:

- Check whether or not the Firewall pre arm bit is set.

Return value:

- FPA: bit setting status (TRUE or FALSE).

Notes:

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

FIREWALL pre arm status

FIREWALL_PRE_ARM_RESET

FIREWALL_PRE_ARM_SET

FIREWALL volatile data segment execution status

FIREWALL_VOLATILEDATA_NOT_EXECUTABLE

FIREWALL_VOLATILEDATA_EXECUTABLE

FIREWALL volatile data segment share status

FIREWALL_VOLATILEDATA_NOT_SHARED

FIREWALL_VOLATILEDATA_SHARED

24 HAL FLASH Generic Driver

24.1 FLASH Firmware driver registers structures

24.1.1 FLASH_EraseInitTypeDef

Data Fields

- ***uint32_t TypeErase***
- ***uint32_t Banks***
- ***uint32_t Page***
- ***uint32_t NbPages***

Field Documentation

- ***uint32_t FLASH_EraseInitTypeDef::TypeErase***
  Mass erase or page erase. This parameter can be a value of ***FLASH_Type_Erase***
- ***uint32_t FLASH_EraseInitTypeDef::Banks***
  Select bank to erase. This parameter must be a value of ***FLASH_Banks*** (FLASH_BANK_BOTH should be used only for mass erase)
- ***uint32_t FLASH_EraseInitTypeDef::Page***
  Initial Flash page to erase when page erase is disabled This parameter must be a value between 0 and (max number of pages in the bank - 1) (eg: 255 for 1MB dual bank)
- ***uint32_t FLASH_EraseInitTypeDef::NbPages***
  Number of pages to be erased. This parameter must be a value between 1 and (max number of pages in the bank - value of initial page)

24.1.2 FLASH_OBProgramInitTypeDef

Data Fields

- ***uint32_t OptionType***
- ***uint32_t WRPArea***
- ***uint32_t WRPStartOffset***
- ***uint32_t WRPEndOffset***
- ***uint32_t RDPLevel***
- ***uint32_t USERType***
- ***uint32_t USERConfig***
- ***uint32_t PCROPConfig***
- ***uint32_t PCROPStartAddr***
- ***uint32_t PCROPEndAddr***

Field Documentation

- ***uint32_t FLASH_OBProgramInitTypeDef::OptionType***
  Option byte to be configured. This parameter can be a combination of the values of ***FLASH_OB_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPArea***
  Write protection area to be programmed (used for OPTIONBYTE_WRP). Only one WRP area could be programmed at the same time. This parameter can be value of ***FLASH_OB_WRP_Area***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPStartOffset***
  Write protection start offset (used for OPTIONBYTE_WRP). This parameter must be a

value between 0 and (max number of pages in the bank - 1) (eg: 25 for 1MB dual bank)

- ***uint32_t FLASH_OBProgramInitTypeDef::WRPEndOffset***
  Write protection end offset (used for OPTIONBYTE_WRP). This parameter must be a value between WRPStartOffset and (max number of pages in the bank - 1)
- ***uint32_t FLASH_OBProgramInitTypeDef::RDPLevel***
  Set the read protection level.. (used for OPTIONBYTE_RDP). This parameter can be a value of ***FLASH_OB_Read_Protection***
- ***uint32_t FLASH_OBProgramInitTypeDef::USERType***
  User option byte(s) to be configured (used for OPTIONBYTE_USER). This parameter can be a combination of ***FLASH_OB_USER_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::USERConfig***
  Value of the user option byte (used for OPTIONBYTE_USER). This parameter can be a combination of ***FLASH_OB_USER_BOR_LEVEL***, ***FLASH_OB_USER_nRST_STOP***, ***FLASH_OB_USER_nRST_STANDBY***, ***FLASH_OB_USER_nRST_SHUTDOWN***, ***FLASH_OB_USER_IWDG_SW***, ***FLASH_OB_USER_IWDG_STOP***, ***FLASH_OB_USER_IWDG_STANDBY***, ***FLASH_OB_USER_WWDG_SW***, ***FLASH_OB_USER_BFB2***, ***FLASH_OB_USER_DUALBANK***, ***FLASH_OB_USER_nBOOT1***, ***FLASH_OB_USER_SRAM2_PE*** and ***FLASH_OB_USER_SRAM2_RST***
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROPConfig***
  Configuration of the PCROP (used for OPTIONBYTE_PCROP). This parameter must be a combination of ***FLASH_Banks*** (except FLASH_BANK_BOTH) and ***FLASH_OB_PCROP_RDP***
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROPStartAddr***
  PCROP Start address (used for OPTIONBYTE_PCROP). This parameter must be a value between begin and end of bank => Be careful of the bank swapping for the address
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROPEndAddr***
  PCROP End address (used for OPTIONBYTE_PCROP). This parameter must be a value between PCROP Start address and end of bank

24.1.3 FLASH_ProcessTypeDef

Data Fields

- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***__IO FLASH_ProcedureTypeDef ProcedureOnGoing***
- ***__IO uint32_t Address***
- ***__IO uint32_t Bank***
- ***__IO uint32_t Page***
- ***__IO uint32_t NbPagesToErase***
- ***__IO FLASH_CacheTypeDef CacheToReactivate***

Field Documentation

- ***HAL_LockTypeDef FLASH_ProcessTypeDef::Lock***
- ***__IO uint32_t FLASH_ProcessTypeDef::ErrorCode***
- ***__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing***
- ***__IO uint32_t FLASH_ProcessTypeDef::Address***
- ***__IO uint32_t FLASH_ProcessTypeDef::Bank***
- ***__IO uint32_t FLASH_ProcessTypeDef::Page***
- ***__IO uint32_t FLASH_ProcessTypeDef::NbPagesToErase***
- ***__IO FLASH_CacheTypeDef FLASH_ProcessTypeDef::CacheToReactivate***

24.2 FLASH Firmware driver API description

24.2.1 FLASH peripheral features

The Flash memory interface manages CPU AHB I-Code and D-Code accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The Flash memory interface accelerates code execution with a system of instruction prefetch and cache lines.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Option bytes programming
- Prefetch on I-Code
- 32 cache lines of 4*64 bits on I-Code
- 8 cache lines of 4*64 bits on D-Code
- Error code correction (ECC): Data in flash are 72-bits word (8 bits added per double word)

24.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32L4xx devices.

1. Flash Memory IO Programming functions:
   - Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   - Program functions: double word and fast program (full row programming)
   - There Two modes of programming:
     - Polling mode using HAL_FLASH_Program() function
     - Interrupt mode using HAL_FLASH_Program_IT() function
2. Interrupts and flags management functions:
   - Handle FLASH interrupts by calling HAL_FLASH_IRQHandler()
   - Callback functions are called when the flash operations are finished: HAL_FLASH_EndOfOperationCallback() when everything is ok, otherwise HAL_FLASH_OperationErrorCallback()
   - Get error flag status by calling HAL_GetError()
3. Option bytes management functions:
   - Lock and Unlock the option bytes using HAL_FLASH_OB_Unlock() and HAL_FLASH_OB_Lock() functions
   - Launch the reload of the option bytes using HAL_FLASH_Launch() function. In this case, a reset is generated

In addition to these functions, this driver includes a set of macros allowing to handle the following operations:

- Set the latency
- Enable/Disable the prefetch buffer
- Enable/Disable the Instruction cache and the Data cache
- Reset the Instruction cache and the Data cache
- Enable/Disable the Flash power-down during low-power run and sleep modes
- Enable/Disable the Flash interrupts
- Monitor the Flash flags status

24.2.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

This section contains the following APIs:

- *HAL_FLASH_Program()*
- *HAL_FLASH_Program_IT()*
- *HAL_FLASH_IRQHandler()*
- *HAL_FLASH_EndOfOperationCallback()*
- *HAL_FLASH_OperationErrorCallback()*

24.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

This section contains the following APIs:

- *HAL_FLASH_Unlock()*
- *HAL_FLASH_Lock()*
- *HAL_FLASH_OB_Unlock()*
- *HAL_FLASH_OB_Lock()*
- *HAL_FLASH_OB_Launch()*

24.2.5 Peripheral Errors functions

This subsection permits to get in run-time Errors of the FLASH peripheral.

This section contains the following APIs:

- *HAL_FLASH_GetError()*

24.2.6 Detailed description of functions

HAL_FLASH_Program

Function name	**HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint64_t Data)**
Function description	Program double word or fast program of a row at a specified address.
Parameters	• **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Program Type • **Address:** specifies the address to be programmed. • **Data:** specifies the data to be programmed This parameter is the data for the double word program and the address where are stored the data for the row fast program
Return values	• **HAL_StatusTypeDef:** HAL Status

HAL_FLASH_Program_IT

Function name	**HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t Data)**
Function description	Program double word or fast program of a row at a specified

address with interrupt enabled.

Parameters	• **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Program Type • **Address:** specifies the address to be programmed. • **Data:** specifies the data to be programmed This parameter is the data for the double word program and the address where are stored the data for the row fast program
Return values	• **HAL:** Status

HAL_FLASH_IRQHandler

Function name	**void HAL_FLASH_IRQHandler (void)**
Function description	Handle FLASH interrupt request.
Return values	• **None:**

HAL_FLASH_EndOfOperationCallback

Function name	**void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)**
Function description	FLASH end of operation interrupt callback.
Parameters	• **ReturnValue:** The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Page Erase: Page which has been erased (if 0xFFFFFFFF, it means that all the selected pages have been erased) Program: Address which was selected for data program
Return values	• **None:**

HAL_FLASH_OperationErrorCallback

Function name	**void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)**
Function description	FLASH operation error interrupt callback.
Parameters	• **ReturnValue:** The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Page Erase: Page number which returned an error Program: Address which was selected for data program
Return values	• **None:**

HAL_FLASH_Unlock

Function name	**HAL_StatusTypeDef HAL_FLASH_Unlock (void)**
Function description	Unlock the FLASH control register access.
Return values	• **HAL:** Status

HAL_FLASH_Lock

Function name	**HAL_StatusTypeDef HAL_FLASH_Lock (void)**
Function description	Lock the FLASH control register access.
Return values	• **HAL:** Status

HAL_FLASH_OB_Unlock

Function name	**HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)**
Function description	Unlock the FLASH Option Bytes Registers access.
Return values	• **HAL:** Status

HAL_FLASH_OB_Lock

Function name	**HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)**
Function description	Lock the FLASH Option Bytes Registers access.
Return values	• **HAL:** Status

HAL_FLASH_OB_Launch

Function name	**HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)**
Function description	Launch the option byte loading.
Return values	• **HAL:** Status

HAL_FLASH_GetError

Function name	**uint32_t HAL_FLASH_GetError (void)**
Function description	Get the specific FLASH error flag.
Return values	• **FLASH_ErrorCode:** The returned value can be: – HAL_FLASH_ERROR_RD: FLASH Read Protection error flag (PCROP) – HAL_FLASH_ERROR_PGS: FLASH Programming Sequence error flag – HAL_FLASH_ERROR_PGP: FLASH Programming Parallelism error flag – HAL_FLASH_ERROR_PGA: FLASH Programming Alignment error flag – HAL_FLASH_ERROR_WRP: FLASH Write protected error flag – HAL_FLASH_ERROR_OPERATION: FLASH operation Error flag – HAL_FLASH_ERROR_NONE: No error set – HAL_FLASH_ERROR_OP: FLASH Operation error – HAL_FLASH_ERROR_PROG: FLASH Programming error – HAL_FLASH_ERROR_WRP: FLASH Write protection error – HAL_FLASH_ERROR_PGA: FLASH Programming alignment error

- HAL_FLASH_ERROR_SIZ: FLASH Size error
- HAL_FLASH_ERROR_PGS: FLASH Programming sequence error
- HAL_FLASH_ERROR_MIS: FLASH Fast programming data miss error
- HAL_FLASH_ERROR_FAST: FLASH Fast programming error
- HAL_FLASH_ERROR_RD: FLASH PCROP read error
- HAL_FLASH_ERROR_OPTV: FLASH Option validity error
- FLASH_FLAG_PEMPTY: FLASH Boot from not programmed flash (apply only for STM32L43x/STM32L44x devices)
- HAL_FLASH_ERROR_ECCD: FLASH two ECC errors have been detected

24.3 FLASH Firmware driver defines

24.3.1 FLASH

FLASH Banks

FLASH_BANK_1	Bank 1
FLASH_BANK_2	Bank 2
FLASH_BANK_BOTH	Bank1 and Bank2

FLASH Error

HAL_FLASH_ERROR_NONE

HAL_FLASH_ERROR_OP

HAL_FLASH_ERROR_PROG

HAL_FLASH_ERROR_WRP

HAL_FLASH_ERROR_PGA

HAL_FLASH_ERROR_SIZ

HAL_FLASH_ERROR_PGS

HAL_FLASH_ERROR_MIS

HAL_FLASH_ERROR_FAST

HAL_FLASH_ERROR_RD

HAL_FLASH_ERROR_OPTV

HAL_FLASH_ERROR_ECCD

HAL_FLASH_ERROR_PEMPTY

FLASH Exported Macros

__HAL_FLASH_SET_LATENCY

Description:

- Set the FLASH Latency.

Parameters:

- __LATENCY__: FLASH

Latency This parameter can be one of the following values:
  - FLASH_LATENCY_0: FLASH Zero wait state
  - FLASH_LATENCY_1: FLASH One wait state
  - FLASH_LATENCY_2: FLASH Two wait states
  - FLASH_LATENCY_3: FLASH Three wait states
  - FLASH_LATENCY_4: FLASH Four wait states

Return value:
- None

__HAL_FLASH_GET_LATENCY

Description:
- Get the FLASH Latency.

Return value:
- FLASH: Latency This parameter can be one of the following values:
  - FLASH_LATENCY_0: FLASH Zero wait state
  - FLASH_LATENCY_1: FLASH One wait state
  - FLASH_LATENCY_2: FLASH Two wait states
  - FLASH_LATENCY_3: FLASH Three wait states
  - FLASH_LATENCY_4: FLASH Four wait states

__HAL_FLASH_PREFETCH_BUFFER_ENABLE

Description:
- Enable the FLASH prefetch buffer.

Return value:
- None

__HAL_FLASH_PREFETCH_BUFFER_DISABLE

Description:
- Disable the FLASH prefetch buffer.

Return value:
- None

__HAL_FLASH_INSTRUCTION_CACHE_ENABLE

Description:
- Enable the FLASH instruction cache.

Return value:

- none

__HAL_FLASH_INSTRUCTION_CACHE_DISABLE	**Description:** • Disable the FLASH instruction cache. **Return value:** • none
__HAL_FLASH_DATA_CACHE_ENABLE	**Description:** • Enable the FLASH data cache. **Return value:** • none
__HAL_FLASH_DATA_CACHE_DISABLE	**Description:** • Disable the FLASH data cache. **Return value:** • none
__HAL_FLASH_INSTRUCTION_CACHE_RESET	**Description:** • Reset the FLASH instruction Cache. **Return value:** • None **Notes:** • This function must be used only when the Instruction Cache is disabled.
__HAL_FLASH_DATA_CACHE_RESET	**Description:** • Reset the FLASH data Cache. **Return value:** • None **Notes:** • This function must be used only when the data Cache is disabled.
__HAL_FLASH_POWER_DOWN_ENABLE	**Notes:** • Writing this bit to 0 this bit, automatically the keys are loss and a new unlock sequence is necessary to re-write it to 1.
__HAL_FLASH_POWER_DOWN_DISABLE	**Notes:** • Writing this bit to 0 this bit,

automatically the keys are loss and a new unlock sequence is necessary to re-write it to 1.

__HAL_FLASH_SLEEP_POWERDOWN_ENABLE

Description:

- Enable the FLASH power down during Low-Power sleep mode.

Return value:

- none

__HAL_FLASH_SLEEP_POWERDOWN_DISABLE

Description:

- Disable the FLASH power down during Low-Power sleep mode.

Return value:

- none

FLASH Flags Definition

FLASH_FLAG_EOP	FLASH End of operation flag
FLASH_FLAG_OPERR	FLASH Operation error flag
FLASH_FLAG_PROGERR	FLASH Programming error flag
FLASH_FLAG_WRPERR	FLASH Write protection error flag
FLASH_FLAG_PGAERR	FLASH Programming alignment error flag
FLASH_FLAG_SIZERR	FLASH Size error flag
FLASH_FLAG_PGSERR	FLASH Programming sequence error flag
FLASH_FLAG_MISERR	FLASH Fast programming data miss error flag
FLASH_FLAG_FASTERR	FLASH Fast programming error flag
FLASH_FLAG_RDERR	FLASH PCROP read error flag
FLASH_FLAG_OPTVERR	FLASH Option validity error flag
FLASH_FLAG_BSY	FLASH Busy flag
FLASH_FLAG_PEMPTY	FLASH Program empty
FLASH_FLAG_ECCC	FLASH ECC correction
FLASH_FLAG_ECCD	FLASH ECC detection
FLASH_FLAG_ALL_ERRORS	

FLASH Interrupts Macros

__HAL_FLASH_ENABLE_IT

Description:

- Enable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
  - FLASH_IT_EOP: End of FLASH Operation

Interrupt
- FLASH_IT_OPERR: Error Interrupt
- FLASH_IT_RDERR: PCROP Read Error Interrupt
- FLASH_IT_ECCC: ECC Correction Interrupt

Return value:

- none

__HAL_FLASH_DISABLE_IT

Description:

- Disable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
  - FLASH_IT_EOP: End of FLASH Operation Interrupt
  - FLASH_IT_OPERR: Error Interrupt
  - FLASH_IT_RDERR: PCROP Read Error Interrupt
  - FLASH_IT_ECCC: ECC Correction Interrupt

Return value:

- none

__HAL_FLASH_GET_FLAG

Description:

- Check whether the specified FLASH flag is set or not.

Parameters:

- __FLAG__: specifies the FLASH flag to check. This parameter can be one of the following values:
  - FLASH_FLAG_EOP: FLASH End of Operation flag
  - FLASH_FLAG_OPERR: FLASH Operation error flag
  - FLASH_FLAG_PROGERR: FLASH Programming error flag
  - FLASH_FLAG_WRPERR: FLASH Write protection error flag
  - FLASH_FLAG_PGAERR: FLASH Programming alignment error flag
  - FLASH_FLAG_SIZERR: FLASH Size error flag
  - FLASH_FLAG_PGSERR: FLASH Programming sequence error flag
  - FLASH_FLAG_MISERR: FLASH Fast programming data miss error flag
  - FLASH_FLAG_FASTERR: FLASH Fast programming error flag
  - FLASH_FLAG_RDERR: FLASH PCROP read error flag
  - FLASH_FLAG_OPTVERR: FLASH Option validity error flag
  - FLASH_FLAG_BSY: FLASH write/erase

operations in progress flag
- FLASH_FLAG_PEMPTY: FLASH Boot from not programmed flash (apply only for STM32L43x/STM32L44x devices)
- FLASH_FLAG_ECCC: FLASH one ECC error has been detected and corrected
- FLASH_FLAG_ECCD: FLASH two ECC errors have been detected

Return value:

- The: new state of FLASH_FLAG (SET or RESET).

__HAL_FLASH_CLEAR_FLAG

Description:

- Clear the FLASH's pending flags.

Parameters:

- __FLAG__: specifies the FLASH flags to clear. This parameter can be any combination of the following values:
  - FLASH_FLAG_EOP: FLASH End of Operation flag
  - FLASH_FLAG_OPERR: FLASH Operation error flag
  - FLASH_FLAG_PROGERR: FLASH Programming error flag
  - FLASH_FLAG_WRPERR: FLASH Write protection error flag
  - FLASH_FLAG_PGAERR: FLASH Programming alignment error flag
  - FLASH_FLAG_SIZERR: FLASH Size error flag
  - FLASH_FLAG_PGSERR: FLASH Programming sequence error flag
  - FLASH_FLAG_MISERR: FLASH Fast programming data miss error flag
  - FLASH_FLAG_FASTERR: FLASH Fast programming error flag
  - FLASH_FLAG_RDERR: FLASH PCROP read error flag
  - FLASH_FLAG_OPTVERR: FLASH Option validity error flag
  - FLASH_FLAG_ECCC: FLASH one ECC error has been detected and corrected
  - FLASH_FLAG_ECCD: FLASH two ECC errors have been detected
  - FLASH_FLAG_ALL_ERRORS: FLASH All errors flags

Return value:

- None

FLASH Interrupts Definition

FLASH_IT_EOP	End of FLASH Operation Interrupt source
FLASH_IT_OPERR	Error Interrupt source

FLASH_IT_RDERR	PCROP Read Error Interrupt source
FLASH_IT_ECCC	ECC Correction Interrupt source

FLASH Keys

FLASH_KEY1	Flash key1
FLASH_KEY2	Flash key2: used with FLASH_KEY1 to unlock the FLASH registers access
FLASH_PDKEY1	Flash power down key1
FLASH_PDKEY2	Flash power down key2: used with FLASH_PDKEY1 to unlock the RUN_PD bit in FLASH_ACR
FLASH_OPTKEY1	Flash option byte key1
FLASH_OPTKEY2	Flash option byte key2: used with FLASH_OPTKEY1 to allow option bytes operations

FLASH Latency

FLASH_LATENCY_0	FLASH Zero wait state
FLASH_LATENCY_1	FLASH One wait state
FLASH_LATENCY_2	FLASH Two wait states
FLASH_LATENCY_3	FLASH Three wait states
FLASH_LATENCY_4	FLASH Four wait states
FLASH_LATENCY_5	FLASH Five wait state
FLASH_LATENCY_6	FLASH Six wait state
FLASH_LATENCY_7	FLASH Seven wait states
FLASH_LATENCY_8	FLASH Eight wait states
FLASH_LATENCY_9	FLASH Nine wait states
FLASH_LATENCY_10	FLASH Ten wait state
FLASH_LATENCY_11	FLASH Eleven wait state
FLASH_LATENCY_12	FLASH Twelve wait states
FLASH_LATENCY_13	FLASH Thirteen wait states
FLASH_LATENCY_14	FLASH Fourteen wait states
FLASH_LATENCY_15	FLASH Fifteen wait states

FLASH Option Bytes PCROP On RDP Level Type

OB_PCROP_RDP_NOT_ERASE	PCROP area is not erased when the RDP level is decreased from Level 1 to Level 0
OB_PCROP_RDP_ERASE	PCROP area is erased when the RDP level is decreased from Level 1 to Level 0 (full mass erase)

FLASH Option Bytes Read Protection

OB_RDP_LEVEL_0	
OB_RDP_LEVEL_1	
OB_RDP_LEVEL_2	Warning: When enabling read protection level 2 it's no more

possible to go back to level 1 or 0

FLASH Option Bytes Type

OPTIONBYTE_WRP	WRP option byte configuration
OPTIONBYTE_RDP	RDP option byte configuration
OPTIONBYTE_USER	USER option byte configuration
OPTIONBYTE_PCROP	PCROP option byte configuration

FLASH Option Bytes User BFB2 Mode

OB_BFB2_DISABLE	Dual-bank boot disable
OB_BFB2_ENABLE	Dual-bank boot enable

FLASH Option Bytes User BOR Level

OB_BOR_LEVEL_0	Reset level threshold is around 1.7V
OB_BOR_LEVEL_1	Reset level threshold is around 2.0V
OB_BOR_LEVEL_2	Reset level threshold is around 2.2V
OB_BOR_LEVEL_3	Reset level threshold is around 2.5V
OB_BOR_LEVEL_4	Reset level threshold is around 2.8V

FLASH Option Bytes User DBANK Type

OB_DBANK_128_BITS	Single-bank with 128-bits data
OB_DBANK_64_BITS	Dual-bank with 64-bits data

FLASH Option Bytes User Dual-bank Type

OB_DUALBANK_SINGLE	1 MB/512 kB Single-bank Flash
OB_DUALBANK_DUAL	1 MB/512 kB Dual-bank Flash

FLASH Option Bytes User IWDG Mode On Standby

OB_IWDG_STDBY_FREEZE	Independent watchdog counter is frozen in Standby mode
OB_IWDG_STDBY_RUN	Independent watchdog counter is running in Standby mode

FLASH Option Bytes User IWDG Mode On Stop

OB_IWDG_STOP_FREEZE	Independent watchdog counter is frozen in Stop mode
OB_IWDG_STOP_RUN	Independent watchdog counter is running in Stop mode

FLASH Option Bytes User IWDG Type

OB_IWDG_HW	Hardware independent watchdog
OB_IWDG_SW	Software independent watchdog

FLASH Option Bytes User BOOT1 Type

OB_BOOT1_SRAM	Embedded SRAM1 is selected as boot space (if BOOT0=1)
OB_BOOT1_SYSTEM	System memory is selected as boot space (if BOOT0=1)

FLASH Option Bytes User Reset On Shutdown

OB_SHUTDOWN_RST	Reset generated when entering the shutdown mode
OB_SHUTDOWN_NORST	No reset generated when entering the shutdown mode

FLASH Option Bytes User Reset On Standby

OB_STANDBY_RST	Reset generated when entering the standby mode
OB_STANDBY_NORST	No reset generated when entering the standby mode

FLASH Option Bytes User Reset On Stop

OB_STOP_RST	Reset generated when entering the stop mode
OB_STOP_NORST	No reset generated when entering the stop mode

FLASH Option Bytes User SRAM2 Parity Check Type

OB_SRAM2_PARITY_ENABLE	SRAM2 parity check enable
OB_SRAM2_PARITY_DISABLE	SRAM2 parity check disable

FLASH Option Bytes User SRAM2 Erase On Reset Type

OB_SRAM2_RST_ERASE	SRAM2 erased when a system reset occurs
OB_SRAM2_RST_NOT_ERASE	SRAM2 is not erased when a system reset occurs

FLASH Option Bytes User Type

OB_USER_BOR_LEV	BOR reset Level
OB_USER_nRST_STOP	Reset generated when entering the stop mode
OB_USER_nRST_STDBY	Reset generated when entering the standby mode
OB_USER_IWDG_SW	Independent watchdog selection
OB_USER_IWDG_STOP	Independent watchdog counter freeze in stop mode
OB_USER_IWDG_STDBY	Independent watchdog counter freeze in standby mode
OB_USER_WWDG_SW	Window watchdog selection
OB_USER_BFB2	Dual-bank boot
OB_USER_DUALBANK	Dual-Bank on 1MB or 512kB Flash memory devices
OB_USER_nBOOT1	Boot configuration
OB_USER_SRAM2_PE	SRAM2 parity check enable
OB_USER_SRAM2_RST	SRAM2 Erase when system reset
OB_USER_nRST_SHDW	Reset generated when entering the shutdown mode
OB_USER_nSWBOOT0	Software BOOT0
OB_USER_nBOOT0	nBOOT0 option bit
OB_USER_DBANK	Single bank with 128-bits data or two banks with 64-bits data

FLASH Option Bytes User WWDG Type

OB_WWDG_HW	Hardware window watchdog
OB_WWDG_SW	Software window watchdog

FLASH WRP Area

OB_WRPAREA_BANK1_AREAA	Flash Bank 1 Area A
OB_WRPAREA_BANK1_AREAB	Flash Bank 1 Area B
OB_WRPAREA_BANK2_AREAA	Flash Bank 2 Area A

OB_WRPAREA_BANK2_AREAB	Flash Bank 2 Area B

FLASH Erase Type

FLASH_TYPEERASE_PAGES	Pages erase only
FLASH_TYPEERASE_MASSERASE	Flash mass erase activation

FLASH Program Type

FLASH_TYPEPROGRAM_DOUBLEWORD	Program a double-word (64-bit) at a specified address.
FLASH_TYPEPROGRAM_FAST	Fast program a 32 row double-word (64-bit) at a specified address. And another 32 row double-word (64-bit) will be programmed
FLASH_TYPEPROGRAM_FAST_AND_LAST	Fast program a 32 row double-word (64-bit) at a specified address. And this is the last 32 row double-word (64-bit) programmed

25 HAL FLASH Extension Driver

25.1 FLASHEx Firmware driver API description

25.1.1 Flash Extended features

Comparing to other previous devices, the FLASH interface for STM32L4xx devices contains the following additional features

- Capacity up to 2 Mbyte with dual bank architecture supporting read-while-write capability (RWW)
- Dual bank memory organization
- PCROP protection for all banks

25.1.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32L4xx devices. It includes

1. Flash Memory Erase functions:
   - Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   - Erase function: Erase page, erase all sectors
   - There are two modes of erase:
     - Polling Mode using HAL_FLASHEx_Erase()
     - Interrupt Mode using HAL_FLASHEx_Erase_IT()
2. Option Bytes Programming function: Use HAL_FLASHEx_OBProgram() to:
   - Set/Reset the write protection
   - Set the Read protection Level
   - Program the user Option Bytes
   - Configure the PCROP protection
3. Get Option Bytes Configuration function: Use HAL_FLASHEx_OBGetConfig() to:
   - Get the value of a write protection area
   - Know if the read protection is activated
   - Get the value of the user Option Bytes
   - Get the value of a PCROP area

25.1.3 Extended programming operation functions

This subsection provides a set of functions allowing to manage the Extended FLASH programming operations Operations.

This section contains the following APIs:

- ***HAL_FLASHEx_Erase()***
- ***HAL_FLASHEx_Erase_IT()***
- ***HAL_FLASHEx_OBProgram()***
- ***HAL_FLASHEx_OBGetConfig()***

25.1.4 Extended specific configuration functions

This subsection provides a set of functions allowing to manage the Extended FLASH specific configurations.

This section contains the following APIs:

- *HAL_FLASHEx_ConfigLVEPin()*

25.1.5 Detailed description of functions

HAL_FLASHEx_Erase

Function name	**HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)**
Function description	Perform a mass erase or erase the specified FLASH memory pages.
Parameters	• **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing. • **PageError:** : pointer to variable that contains the configuration information on faulty page in case of error (0xFFFFFFFF means that all the pages have been correctly erased)
Return values	• **HAL:** Status

HAL_FLASHEx_Erase_IT

Function name	**HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)**
Function description	Perform a mass erase or erase the specified FLASH memory pages with interrupt enabled.
Parameters	• **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
Return values	• **HAL:** Status

HAL_FLASHEx_OBProgram

Function name	**HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)**
Function description	Program Option bytes.
Parameters	• **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	• **HAL:** Status

HAL_FLASHEx_OBGetConfig

Function name	**void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)**
Function description	Get the Option bytes configuration.
Parameters	• **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information.
Return values	• **None:**
Notes	• The fields pOBInit->WRPArea and pOBInit->PCROPConfig should indicate which area is requested for the WRP and

PCROP, else no information will be returned

HAL_FLASHEx_ConfigLVEPin

Function name	**HAL_StatusTypeDef HAL_FLASHEx_ConfigLVEPin (uint32_t ConfigLVE)**
Function description	Configuration of the LVE pin of the Flash (managed by power controller or forced to low in order to use an external SMPS)
Parameters	• **ConfigLVE:** Configuration of the LVE pin, This parameter can be one of the following values: – FLASH_LVE_PIN_CTRL: LVE FLASH pin controlled by power controller – FLASH_LVE_PIN_FORCED: LVE FLASH pin enforced to low (external SMPS used)
Return values	• **HAL:** Status
Notes	• Before enforcing the LVE pin to low, the SOC should be in low voltage range 2 and the voltage VDD12 should be higher than 1.08V and SMPS is ON.

25.2 FLASHEx Firmware driver defines

25.2.1 FLASHEx

FLASHEx LVE pin configuration

FLASH_LVE_PIN_CTRL	LVE FLASH pin controlled by power controller
FLASH_LVE_PIN_FORCED	LVE FLASH pin enforced to low (external SMPS used)

26 HAL FLASH__RAMFUNC Generic Driver

26.1 FLASH__RAMFUNC Firmware driver API description

26.1.1 Flash RAM functions

Arm Compiler

RAM functions are defined using the toolchain options. Functions that are executed in RAM should reside in a separate source module. Using the 'Options for File' dialog you can simply change the 'Code / Const' area of a module to a memory space in physical RAM. Available memory areas are declared in the 'Target' tab of the Options for Target' dialog.

ICCARM Compiler

RAM functions are defined using a specific toolchain keyword "__ramfunc".

GNU Compiler

RAM functions are defined using a specific toolchain attribute "__attribute__((section(".RamFunc")))".

26.1.2 ramfunc functions

This subsection provides a set of functions that should be executed from RAM.

This section contains the following APIs:

- *HAL_FLASHEx_EnableRunPowerDown()*
- *HAL_FLASHEx_DisableRunPowerDown()*
- *HAL_FLASHEx_OB_DBankConfig()*

26.1.3 Detailed description of functions

HAL_FLASHEx_EnableRunPowerDown

Function name	**__RAM_FUNC HAL_FLASHEx_EnableRunPowerDown (void)**
Function description	Enable the Power down in Run Mode.
Return values	• **None:**
Notes	• This function should be called and executed from SRAM memory

HAL_FLASHEx_DisableRunPowerDown

Function name	**__RAM_FUNC HAL_FLASHEx_DisableRunPowerDown (void)**
Function description	Disable the Power down in Run Mode.
Return values	• **None:**
Notes	• This function should be called and executed from SRAM memory

HAL_FLASHEx_OB_DBankConfig

Function name	**__RAM_FUNC HAL_FLASHEx_OB_DBankConfig (uint32_t DBankConfig)**
Function description	Program the FLASH DBANK User Option Byte.
Parameters	• **DBankConfig:** The FLASH DBANK User Option Byte value. This parameter can be one of the following values: – OB_DBANK_128_BITS: Single-bank with 128-bits data – OB_DBANK_64_BITS: Dual-bank with 64-bits data
Return values	• **HAL:** status
Notes	• To configure the user option bytes, the option lock bit OPTLOCK must be cleared with the call of the HAL_FLASH_OB_Unlock() function. • To modify the DBANK option byte, no PCROP region should be defined. To deactivate PCROP, user should perform RDP changing

27 HAL GFXMMU Generic Driver

27.1 GFXMMU Firmware driver registers structures

27.1.1 GFXMMU_BuffersTypeDef

Data Fields

- ***uint32_t Buf0Address***
- ***uint32_t Buf1Address***
- ***uint32_t Buf2Address***
- ***uint32_t Buf3Address***

Field Documentation

- ***uint32_t GFXMMU_BuffersTypeDef::Buf0Address***
  Physical address of buffer 0.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf1Address***
  Physical address of buffer 1.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf2Address***
  Physical address of buffer 2.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf3Address***
  Physical address of buffer 3.

27.1.2 GFXMMU_InterruptsTypeDef

Data Fields

- ***FunctionalState Activation***
- ***uint32_t UsedInterrupts***

Field Documentation

- ***FunctionalState GFXMMU_InterruptsTypeDef::Activation***
  Interrupts enable/disable
- ***uint32_t GFXMMU_InterruptsTypeDef::UsedInterrupts***
  Interrupts used. This parameter can be a values combination of ***GFXMMU_Interrupts***.
  Note:: Usefull only when interrupts are enabled.

27.1.3 GFXMMU_InitTypeDef

Data Fields

- ***uint32_t BlocksPerLine***
- ***uint32_t DefaultValue***
- ***GFXMMU_BuffersTypeDef Buffers***
- ***GFXMMU_InterruptsTypeDef Interrupts***

Field Documentation

- ***uint32_t GFXMMU_InitTypeDef::BlocksPerLine***
  Number of blocks of 16 bytes per line. This parameter can be a value of ***GFXMMU_BlocksPerLine***.
- ***uint32_t GFXMMU_InitTypeDef::DefaultValue***
  Value returned when virtual memory location not physically mapped.
- ***GFXMMU_BuffersTypeDef GFXMMU_InitTypeDef::Buffers***
  Physical buffers addresses.

- ***GFXMMU_InterruptsTypeDef GFXMMU_InitTypeDef::Interrupts***
  Interrupts parameters.

27.1.4 GFXMMU_HandleTypeDef

Data Fields

- ***GFXMMU_TypeDef * Instance***
- ***GFXMMU_InitTypeDef Init***
- ***HAL_GFXMMU_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***GFXMMU_TypeDef* GFXMMU_HandleTypeDef::Instance***
  GFXMMU instance
- ***GFXMMU_InitTypeDef GFXMMU_HandleTypeDef::Init***
  GFXMMU init parameters
- ***HAL_GFXMMU_StateTypeDef GFXMMU_HandleTypeDef::State***
  GFXMMU state
- ***__IO uint32_t GFXMMU_HandleTypeDef::ErrorCode***
  GFXMMU error code

27.1.5 GFXMMU_LutLineTypeDef

Data Fields

- ***uint32_t LineNumber***
- ***uint32_t LineStatus***
- ***uint32_t FirstVisibleBlock***
- ***uint32_t LastVisibleBlock***
- ***int32_t LineOffset***

Field Documentation

- ***uint32_t GFXMMU_LutLineTypeDef::LineNumber***
  LUT line number. This parameter must be a number between Min_Data = 0 and Max_Data = 1023.
- ***uint32_t GFXMMU_LutLineTypeDef::LineStatus***
  LUT line enable/disable. This parameter can be a value of ***GFXMMU_LutLineStatus***.
- ***uint32_t GFXMMU_LutLineTypeDef::FirstVisibleBlock***
  First visible block on this line. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- ***uint32_t GFXMMU_LutLineTypeDef::LastVisibleBlock***
  Last visible block on this line. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- ***int32_t GFXMMU_LutLineTypeDef::LineOffset***
  Offset of block 0 of the current line in physical buffer. This parameter must be a number between Min_Data = -4080 and Max_Data = 4190208.
  Note:: Line offset has to be computed with the following formula: LineOffset = [(Blocks already used) - (1st visible block)]*BlockSize.

27.2 GFXMMU Firmware driver API description

27.2.1 How to use this driver

Initialization

1. As prerequisite, fill in the HAL_GFXMMU_MspInit():
   - Enable GFXMMU clock interface with __HAL_RCC_GFXMMU_CLK_ENABLE().
   - If interrupts are used, enable and configure GFXMMU global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Configure the number of blocks per line, default value, physical buffer addresses and interrupts using the HAL_GFXMMU_Init() function.

LUT configuration

1. Use HAL_GFXMMU_DisableLutLines() to deactivate all LUT lines (or a range of lines).
2. Use HAL_GFXMMU_ConfigLut() to copy LUT from flash to look up RAM.
3. Use HAL_GFXMMU_ConfigLutLine() to configure one line of LUT.

Modify physical buffer adresses

1. Use HAL_GFXMMU_ModifyBuffers() to modify physical buffer addresses.

Error management

1. If interrupts are used, HAL_GFXMMU_IRQHandler() will be called when an error occurs. This function will call HAL_GFXMMU_ErrorCallback(). Use HAL_GFXMMU_GetError() to get the error code.

De-initialization

1. As prerequisite, fill in the HAL_GFXMMU_MspDeInit():
   - Disable GFXMMU clock interface with __HAL_RCC_GFXMMU_CLK_ENABLE().
   - If interrupts has been used, disable GFXMMU global interrupt with HAL_NVIC_DisableIRQ().
2. De-initialize GFXMMU using the HAL_GFXMMU_DeInit() function.

27.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the GFXMMU.
- De-initialize the GFXMMU.

This section contains the following APIs:

- ***HAL_GFXMMU_Init()***
- ***HAL_GFXMMU_DeInit()***
- ***HAL_GFXMMU_MspInit()***
- ***HAL_GFXMMU_MspDeInit()***

27.2.3 Operation functions

This section provides functions allowing to:

- Configure LUT.

- Modify physical buffer adresses.
- Manage error.

This section contains the following APIs:

- ***HAL_GFXMMU_ConfigLut()***
- ***HAL_GFXMMU_DisableLutLines()***
- ***HAL_GFXMMU_ConfigLutLine()***
- ***HAL_GFXMMU_ModifyBuffers()***
- ***HAL_GFXMMU_IRQHandler()***
- ***HAL_GFXMMU_ErrorCallback()***

27.2.4 State functions

This section provides functions allowing to:

- Get GFXMMU handle state.
- Get GFXMMU error code.

This section contains the following APIs:

- ***HAL_GFXMMU_GetState()***
- ***HAL_GFXMMU_GetError()***

27.2.5 Detailed description of functions

HAL_GFXMMU_Init

Function name	**HAL_StatusTypeDef HAL_GFXMMU_Init (GFXMMU_HandleTypeDef * hgfxmmu)**
Function description	Initialize the GFXMMU according to the specified parameters in the GFXMMU_InitTypeDef structure and initialize the associated handle.
Parameters	• **hgfxmmu:** : GFXMMU handle.
Return values	• **HAL:** status.

HAL_GFXMMU_DeInit

Function name	**HAL_StatusTypeDef HAL_GFXMMU_DeInit (GFXMMU_HandleTypeDef * hgfxmmu)**
Function description	De-initialize the GFXMMU.
Parameters	• **hgfxmmu:** : GFXMMU handle.
Return values	• **HAL:** status.

HAL_GFXMMU_MspInit

Function name	**void HAL_GFXMMU_MspInit (GFXMMU_HandleTypeDef * hgfxmmu)**
Function description	Initialize the GFXMMU MSP.
Parameters	• **hgfxmmu:** : GFXMMU handle.
Return values	• **None.:**

HAL_GFXMMU_MspDeInit

Function name	**void HAL_GFXMMU_MspDeInit (GFXMMU_HandleTypeDef * hgfxmmu)**
Function description	De-initialize the GFXMMU MSP.
Parameters	• **hgfxmmu:** : GFXMMU handle.
Return values	• **None.:**

HAL_GFXMMU_ConfigLut

Function name	**HAL_StatusTypeDef HAL_GFXMMU_ConfigLut (GFXMMU_HandleTypeDef * hgfxmmu, uint32_t FirstLine, uint32_t LinesNumber, uint32_t Address)**
Function description	This function allows to copy LUT from flash to look up RAM.
Parameters	• **hgfxmmu:** : GFXMMU handle. • **FirstLine:** : First line enabled on LUT. This parameter must be a number between Min_Data = 0 and Max_Data = 1023. • **LinesNumber:** : Number of lines enabled on LUT. This parameter must be a number between Min_Data = 1 and Max_Data = 1024. • **Address:** : Start address of LUT in flash.
Return values	• **HAL:** status.

HAL_GFXMMU_DisableLutLines

Function name	**HAL_StatusTypeDef HAL_GFXMMU_DisableLutLines (GFXMMU_HandleTypeDef * hgfxmmu, uint32_t FirstLine, uint32_t LinesNumber)**
Function description	This function allows to disable a range of LUT lines.
Parameters	• **hgfxmmu:** : GFXMMU handle. • **FirstLine:** : First line to disable on LUT. This parameter must be a number between Min_Data = 0 and Max_Data = 1023. • **LinesNumber:** : Number of lines to disable on LUT. This parameter must be a number between Min_Data = 1 and Max_Data = 1024.
Return values	• **HAL:** status.

HAL_GFXMMU_ConfigLutLine

Function name	**HAL_StatusTypeDef HAL_GFXMMU_ConfigLutLine (GFXMMU_HandleTypeDef * hgfxmmu, GFXMMU_LutLineTypeDef * lutLine)**
Function description	This function allows to configure one line of LUT.
Parameters	• **hgfxmmu:** : GFXMMU handle. • **lutLine:** : LUT line parameters.
Return values	• **HAL:** status.

HAL_GFXMMU_ModifyBuffers

Function name	**HAL_StatusTypeDef HAL_GFXMMU_ModifyBuffers (GFXMMU_HandleTypeDef * hgfxmmu, GFXMMU_BuffersTypeDef * Buffers)**
Function description	This function allows to modify physical buffer addresses.
Parameters	• **hgfxmmu:** : GFXMMU handle. • **Buffers:** : Buffers parameters.
Return values	• **HAL:** status.

HAL_GFXMMU_IRQHandler

Function name	**void HAL_GFXMMU_IRQHandler (GFXMMU_HandleTypeDef * hgfxmmu)**
Function description	This function handles the GFXMMU interrupts.
Parameters	• **hgfxmmu:** : GFXMMU handle.
Return values	• **None.:**

HAL_GFXMMU_ErrorCallback

Function name	**void HAL_GFXMMU_ErrorCallback (GFXMMU_HandleTypeDef * hgfxmmu)**
Function description	Error callback.
Parameters	• **hgfxmmu:** : GFXMMU handle.
Return values	• **None.:**

HAL_GFXMMU_GetState

Function name	**HAL_GFXMMU_StateTypeDef HAL_GFXMMU_GetState (GFXMMU_HandleTypeDef * hgfxmmu)**
Function description	This function allows to get the current GFXMMU handle state.
Parameters	• **hgfxmmu:** : GFXMMU handle.
Return values	• **GFXMMU:** state.

HAL_GFXMMU_GetError

Function name	**uint32_t HAL_GFXMMU_GetError (GFXMMU_HandleTypeDef * hgfxmmu)**
Function description	This function allows to get the current GFXMMU error code.
Parameters	• **hgfxmmu:** : GFXMMU handle.
Return values	• **GFXMMU:** error code.

27.3 GFXMMU Firmware driver defines

27.3.1 GFXMMU

GFXMMU blocks per line

GFXMMU_256BLOCKS	256 blocks of 16 bytes per line
GFXMMU_192BLOCKS	192 blocks of 16 bytes per line

GFXMMU Error Code

GFXMMU_ERROR_NONE	No error
GFXMMU_ERROR_BUFFER0_OVERFLOW	Buffer 0 overflow
GFXMMU_ERROR_BUFFER1_OVERFLOW	Buffer 1 overflow
GFXMMU_ERROR_BUFFER2_OVERFLOW	Buffer 2 overflow
GFXMMU_ERROR_BUFFER3_OVERFLOW	Buffer 3 overflow
GFXMMU_ERROR_AHB_MASTER	AHB master error

GFXMMU Exported Macros

__HAL_GFXMMU_RESET_HANDLE_STATE

Description:

- Reset GFXMMU handle state.

Parameters:

- __HANDLE__: GFXMMU handle.

Return value:

- None

GFXMMU interrupts

GFXMMU_AHB_MASTER_ERROR_IT	AHB master error interrupt
GFXMMU_BUFFER0_OVERFLOW_IT	Buffer 0 overflow interrupt
GFXMMU_BUFFER1_OVERFLOW_IT	Buffer 1 overflow interrupt
GFXMMU_BUFFER2_OVERFLOW_IT	Buffer 2 overflow interrupt
GFXMMU_BUFFER3_OVERFLOW_IT	Buffer 3 overflow interrupt

GFXMMU LUT line status

GFXMMU_LUT_LINE_DISABLE	LUT line disabled
GFXMMU_LUT_LINE_ENABLE	LUT line enabled

28 HAL GPIO Generic Driver

28.1 GPIO Firmware driver registers structures

28.1.1 GPIO_InitTypeDef

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Pull***
- ***uint32_t Speed***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_pins***
- ***uint32_t GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_mode***
- ***uint32_t GPIO_InitTypeDef::Pull***
  Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of ***GPIO_pull***
- ***uint32_t GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_speed***
- ***uint32_t GPIO_InitTypeDef::Alternate***
  Peripheral to be connected to the selected pins This parameter can be a value of ***GPIOEx_Alternate_function_selection***

28.2 GPIO Firmware driver API description

28.2.1 GPIO Peripheral features

- Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:
  - Input mode
  - Analog mode
  - Output mode
  - Alternate function mode
  - External interrupt/event lines
- During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.
- All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.
- In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.
- The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.

- All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.
- The external interrupt/event controller consists of up to 39 edge detectors (16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

28.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function: __HAL_RCC_GPIOx_CLK_ENABLE().
2. Configure the GPIO pin(s) using HAL_GPIO_Init().
   - Configure the IO mode using "Mode" member from GPIO_InitTypeDef structure
   - Activate Pull-up, Pull-down resistor using "Pull" member from GPIO_InitTypeDef structure.
   - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from GPIO_InitTypeDef structure.
   - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from GPIO_InitTypeDef structure.
   - Analog mode is required when a pin is to be used as ADC channel or DAC output.
   - In case of external interrupt/event selection the "Mode" member from GPIO_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL_NVIC_SetPriority() and enable it using HAL_NVIC_EnableIRQ().
4. To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
5. To set/reset the level of a pin configured in output mode use HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
6. To lock pin configuration until next reset use HAL_GPIO_LockPin().
7. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
8. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
9. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

28.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_GPIO_Init()***
- ***HAL_GPIO_DeInit()***

28.2.4 IO operation functions

This section contains the following APIs:

- ***HAL_GPIO_ReadPin()***
- ***HAL_GPIO_WritePin()***
- ***HAL_GPIO_TogglePin()***
- ***HAL_GPIO_LockPin()***
- ***HAL_GPIO_EXTI_IRQHandler()***

- *HAL_GPIO_EXTI_Callback()*

28.2.5 Detailed description of functions

HAL_GPIO_Init

Function name	**void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)**
Function description	Initialize the GPIOx peripheral according to the specified parameters in the GPIO_Init.
Parameters	• **GPIOx:** where x can be (A..H) to select the GPIO peripheral for STM32L4 family • **GPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	• **None:**

HAL_GPIO_DeInit

Function name	**void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)**
Function description	De-initialize the GPIOx peripheral registers to their default reset values.
Parameters	• **GPIOx:** where x can be (A..H) to select the GPIO peripheral for STM32L4 family • **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).
Return values	• **None:**

HAL_GPIO_ReadPin

Function name	**GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function description	Read the specified input port pin.
Parameters	• **GPIOx:** where x can be (A..H) to select the GPIO peripheral for STM32L4 family • **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).
Return values	• **The:** input port pin value.

HAL_GPIO_WritePin

Function name	**void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)**
Function description	Set or clear the selected data port bit.
Parameters	• **GPIOx:** where x can be (A..H) to select the GPIO peripheral for STM32L4 family • **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).

- **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values:
  – GPIO_PIN_RESET: to clear the port pin
  – GPIO_PIN_SET: to set the port pin

Return values	• **None:**
Notes	• This function uses GPIOx_BSRR and GPIOx_BRR registers to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

HAL_GPIO_TogglePin

Function name	**void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function description	Toggle the specified GPIO pin.
Parameters	• **GPIOx:** where x can be (A..H) to select the GPIO peripheral for STM32L4 family • **GPIO_Pin:** specifies the pin to be toggled.
Return values	• **None:**

HAL_GPIO_LockPin

Function name	**HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function description	Lock GPIO Pins configuration registers.
Parameters	• **GPIOx:** where x can be (A..H) to select the GPIO peripheral for STM32L4 family • **GPIO_Pin:** specifies the port bits to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
Return values	• **None:**
Notes	• The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH. • The configuration of the locked GPIO pins can no longer be modified until the next reset.

HAL_GPIO_EXTI_IRQHandler

Function name	**void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)**
Function description	Handle EXTI interrupt request.
Parameters	• **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.
Return values	• **None:**

HAL_GPIO_EXTI_Callback

Function name	**void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)**
Function description	EXTI line detection callback.
Parameters	• **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.
Return values	• **None:**

28.3 GPIO Firmware driver defines

28.3.1 GPIO

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_FLAG

Description:

- Check whether the specified EXTI line flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line flag to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_FLAG

Description:

- Clear the EXTI's line pending flags.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines flags to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_IT

Description:

- Check whether the specified EXTI line is asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_IT

Description:

- Clear the EXTI's line pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- None

GPIO mode

GPIO_MODE_INPUT	Input Floating Mode
GPIO_MODE_OUTPUT_PP	Output Push Pull Mode
GPIO_MODE_OUTPUT_OD	Output Open Drain Mode
GPIO_MODE_AF_PP	Alternate Function Push Pull Mode
GPIO_MODE_AF_OD	Alternate Function Open Drain Mode
GPIO_MODE_ANALOG	Analog Mode
GPIO_MODE_ANALOG_ADC_CONTROL	Analog Mode for ADC conversion
GPIO_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
GPIO_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
GPIO_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
GPIO_MODE_EVT_RISING	External Event Mode with Rising edge trigger detection
GPIO_MODE_EVT_FALLING	External Event Mode with Falling edge trigger detection
GPIO_MODE_EVT_RISING_FALLING	External Event Mode with Rising/Falling edge trigger detection

GPIO pins

GPIO_PIN_0

GPIO_PIN_1

GPIO_PIN_2

GPIO_PIN_3

GPIO_PIN_4

GPIO_PIN_5

GPIO_PIN_6

GPIO_PIN_7

GPIO_PIN_8

GPIO_PIN_9

GPIO_PIN_10

GPIO_PIN_11

GPIO_PIN_12

GPIO_PIN_13

GPIO_PIN_14

GPIO_PIN_15

GPIO_PIN_All

GPIO_PIN_MASK

GPIO pull

GPIO_NOPULL	No Pull-up or Pull-down activation
GPIO_PULLUP	Pull-up activation
GPIO_PULLDOWN	Pull-down activation

GPIO speed

GPIO_SPEED_FREQ_LOW	range up to 5 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_MEDIUM	range 5 MHz to 25 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_HIGH	range 25 MHz to 50 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_VERY_HIGH	range 50 MHz to 80 MHz, please refer to the product datasheet

29 HAL GPIO Extension Driver

29.1 GPIOEx Firmware driver defines

29.1.1 GPIOEx

GPIOEx Alternate function selection

GPIO_AF0_RTC_50Hz

GPIO_AF0_MCO

GPIO_AF0_SWJ

GPIO_AF0_TRACE

GPIO_AF1_TIM1

GPIO_AF1_TIM2

GPIO_AF1_TIM5

GPIO_AF1_TIM8

GPIO_AF1_LPTIM1

GPIO_AF1_IR

GPIO_AF2_TIM1

GPIO_AF2_TIM2

GPIO_AF2_TIM3

GPIO_AF2_TIM4

GPIO_AF2_TIM5

GPIO_AF3_I2C4

GPIO_AF3_OCTOSPIM_P1

GPIO_AF3_SAI1

GPIO_AF3_SPI2

GPIO_AF3_TIM1_COMP1

GPIO_AF3_TIM1_COMP2

GPIO_AF3_TIM8

GPIO_AF3_TIM8_COMP1

GPIO_AF3_TIM8_COMP2

GPIO_AF3_USART2

GPIO_AF4_I2C1

GPIO_AF4_I2C2

GPIO_AF4_I2C3

GPIO_AF4_I2C4

GPIO_AF4_DCMI

GPIO_AF5_DCMI

GPIO_AF5_DFSDM1

GPIO_AF5_I2C4

GPIO_AF5_OCTOSPIM_P1

GPIO_AF5_OCTOSPIM_P2

GPIO_AF5_SPI1

GPIO_AF5_SPI2

GPIO_AF5_SPI3

GPIO_AF6_DFSDM1

GPIO_AF6_I2C3

GPIO_AF6_SPI3

GPIO_AF7_USART1

GPIO_AF7_USART2

GPIO_AF7_USART3

GPIO_AF8_LPUART1

GPIO_AF8_SDMMC1

GPIO_AF8_UART4

GPIO_AF8_UART5

GPIO_AF9_CAN1

GPIO_AF9_LTDC

GPIO_AF9_TSC

GPIO_AF10_DCMI

GPIO_AF10_OCTOSPIM_P1

GPIO_AF10_OCTOSPIM_P2

GPIO_AF10_OTG_FS

GPIO_AF11_DSI

GPIO_AF11_LTDC

GPIO_AF12_COMP1

GPIO_AF12_COMP2

GPIO_AF12_DSI

GPIO_AF12_FMC

GPIO_AF12_SDMMC1

GPIO_AF12_TIM1_COMP1

GPIO_AF12_TIM1_COMP2

GPIO_AF12_TIM8_COMP2

GPIO_AF13_SAI1

GPIO_AF13_SAI2

GPIO_AF13_TIM8_COMP1

GPIO_AF14_TIM15

GPIO_AF14_TIM16

GPIO_AF14_TIM17

GPIO_AF14_LPTIM2

GPIO_AF14_TIM8_COMP2

GPIO_AF15_EVENTOUT

IS_GPIO_AF

GPIOEx_Get Port Index

GPIO_GET_INDEX

30 HAL HASH Generic Driver

30.1 HASH Firmware driver registers structures

30.1.1 HASH_InitTypeDef

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint8_t * pKey***

Field Documentation

- ***uint32_t HASH_InitTypeDef::DataType***
  32-bit data, 16-bit data, 8-bit data or 1-bit data. This parameter can be a value of ***HASH_Data_Type***.
- ***uint32_t HASH_InitTypeDef::KeySize***
  The key size is used only in HMAC operation.
- ***uint8_t* HASH_InitTypeDef::pKey***
  The key is used only in HMAC operation.

30.1.2 HASH_HandleTypeDef

Data Fields

- ***HASH_InitTypeDef Init***
- ***uint8_t * pHashInBuffPtr***
- ***uint8_t * pHashOutBuffPtr***
- ***uint8_t * pHashKeyBuffPtr***
- ***uint8_t * pHashMsgBuffPtr***
- ***uint32_t HashBuffSize***
- ***__IO uint32_t HashInCount***
- ***__IO uint32_t HashITCounter***
- ***__IO uint32_t HashKeyCount***
- ***HAL_StatusTypeDef Status***
- ***HAL_HASH_PhaseTypeDef Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_HASH_StateTypeDef State***
- ***HAL_HASH_SuspendTypeDef SuspendRequest***
- ***FlagStatus DigestCalculationDisable***
- ***__IO uint32_t NbWordsAlreadyPushed***

Field Documentation

- ***HASH_InitTypeDef HASH_HandleTypeDef::Init***
  HASH required parameters
- ***uint8_t* HASH_HandleTypeDef::pHashInBuffPtr***
  Pointer to input buffer
- ***uint8_t* HASH_HandleTypeDef::pHashOutBuffPtr***
  Pointer to output buffer (digest)
- ***uint8_t* HASH_HandleTypeDef::pHashKeyBuffPtr***
  Pointer to key buffer (HMAC only)

- ***uint8_t* HASH_HandleTypeDef::pHashMsgBuffPtr***
  Pointer to message buffer (HMAC only)
- ***uint32_t HASH_HandleTypeDef::HashBuffSize***
  Size of buffer to be processed
- ***__IO uint32_t HASH_HandleTypeDef::HashInCount***
  Counter of inputted data
- ***__IO uint32_t HASH_HandleTypeDef::HashITCounter***
  Counter of issued interrupts
- ***__IO uint32_t HASH_HandleTypeDef::HashKeyCount***
  Counter for Key inputted data (HMAC only)
- ***HAL_StatusTypeDef HASH_HandleTypeDef::Status***
  HASH peripheral status
- ***HAL_HASH_PhaseTypeDef HASH_HandleTypeDef::Phase***
  HASH peripheral phase
- ***DMA_HandleTypeDef* HASH_HandleTypeDef::hdmain***
  HASH In DMA Handle parameters
- ***HAL_LockTypeDef HASH_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_HASH_StateTypeDef HASH_HandleTypeDef::State***
  HASH peripheral state
- ***HAL_HASH_SuspendTypeDef HASH_HandleTypeDef::SuspendRequest***
  HASH peripheral suspension request flag
- ***FlagStatus HASH_HandleTypeDef::DigestCalculationDisable***
  Digest calculation phase skip (MDMAT bit control) for multi-buffers DMA-based HMAC computation
- ***__IO uint32_t HASH_HandleTypeDef::NbWordsAlreadyPushed***
  Numbers of words already pushed in FIFO before inputting new block

30.2 HASH Firmware driver API description

30.2.1 How to use this driver

The HASH HAL driver can be used as follows:

1. Initialize the HASH low level resources by implementing the HAL_HASH_MspInit():
   a. Enable the HASH interface clock using __HASH_CLK_ENABLE()
   b. When resorting to interrupt-based APIs (e.g. HAL_HASH_xxx_Start_IT())
      – Configure the HASH interrupt priority using HAL_NVIC_SetPriority()
      – Enable the HASH IRQ handler using HAL_NVIC_EnableIRQ()
      – In HASH IRQ handler, call HAL_HASH_IRQHandler() API
   c. When resorting to DMA-based APIs (e.g. HAL_HASH_xxx_Start_DMA())
      – Enable the DMAx interface clock using __DMAx_CLK_ENABLE()
      – Configure and enable one DMA stream to manage data transfer from memory to peripheral (input stream). Managing data transfer from peripheral to memory can be performed only using CPU.
      – Associate the initialized DMA handle to the HASH DMA handle using __HAL_LINKDMA()
      – Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Stream: use HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HASH HAL using HAL_HASH_Init(). This function:
   a. resorts to HAL_HASH_MspInit() for low-level initialization,
   b. configures the data type: 1-bit, 8-bit, 16-bit or 32-bit.
3. Three processing schemes are available:

a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished, e.g. HAL_HASH_xxx_Start() for HASH or HAL_HMAC_xxx_Start() for HMAC
b. Interrupt mode: processing APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_HASH_xxx_Start_IT() for HASH or HAL_HMAC_xxx_Start_IT() for HMAC
c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA, e.g. HAL_HASH_xxx_Start_DMA() for HASH or HAL_HMAC_xxx_Start_DMA() for HMAC. Note that in DMA mode, a call to HAL_HASH_xxx_Finish() is then required to retrieve the digest.

4. When the processing function is called after HAL_HASH_Init(), the HASH peripheral is initialized and processes the buffer fed in input. When the input data have all been fed to the IP, the digest computation can start.
5. Multi-buffer processing is possible in polling and DMA mode.
   a. In polling mode, only multi-buffer HASH processing is possible. API HAL_HASH_xxx_Accumulate() must be called for each input buffer, except for the last one. User must resort to HAL_HASH_xxx_Start() to enter the last one and retrieve as well the computed digest.
   b. In DMA mode, multi-buffer HASH and HMAC processing are possible.
      – HASH processing: once initialization is done, MDMAT bit must be set thru __HAL_HASH_SET_MDMAT() macro. From that point, each buffer can be fed to the IP thru HAL_HASH_xxx_Start_DMA() API. Before entering the last buffer, reset the MDMAT bit with __HAL_HASH_RESET_MDMAT() macro then wrap-up the HASH processing in feeding the last input buffer thru the same API HAL_HASH_xxx_Start_DMA(). The digest can then be retrieved with a call to API HAL_HASH_xxx_Finish().
      – HMAC processing (requires to resort to extended functions): after initialization, the key and the first input buffer are entered in the IP with the API HAL_HMACEx_xxx_Step1_2_DMA(). This carries out HMAC step 1 and starts step 2. The following buffers are next entered with the API HAL_HMACEx_xxx_Step2_DMA(). At this point, the HMAC processing is still carrying out step 2. Then, step 2 for the last input buffer and step 3 are carried out by a single call to HAL_HMACEx_xxx_Step2_3_DMA(). The digest can finally be retrieved with a call to API HAL_HASH_xxx_Finish().
6. Context swapping.
   a. Two APIs are available to suspend HASH or HMAC processing:
      – HAL_HASH_SwFeed_ProcessSuspend() when data are entered by software (polling or IT mode),
      – HAL_HASH_DMAFeed_ProcessSuspend() when data are entered by DMA.
   b. When HASH or HMAC processing is suspended, HAL_HASH_ContextSaving() allows to save in memory the IP context. This context can be restored afterwards to resume the HASH processing thanks to HAL_HASH_ContextRestoring().
   c. Once the HASH IP has been restored to the same configuration as that at suspension time, processing can be restarted with the same API call (same API, same handle, same parameters) as done before the suspension. Relevant parameters to restart at the proper location are internally saved in the HASH handle.
7. Call HAL_HASH_DeInit() to deinitialize the HASH peripheral.

30.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the HASH according to the specified parameters in the HASH_InitTypeDef and create the associated handle

- DeInitialize the HASH peripheral
- Initialize the HASH MCU Specific Package (MSP)
- DeInitialize the HASH MSP

This section provides as well call back functions definitions for user code to manage:

- Input data transfer to IP completion
- Calculated digest retrieval completion
- Error management

This section contains the following APIs:

- ***HAL_HASH_Init()***
- ***HAL_HASH_DeInit()***
- ***HAL_HASH_MspInit()***
- ***HAL_HASH_MspDeInit()***
- ***HAL_HASH_InCpltCallback()***
- ***HAL_HASH_DgstCpltCallback()***
- ***HAL_HASH_ErrorCallback()***

30.2.3 Polling mode HASH processing functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- MD5
  - HAL_HASH_MD5_Start()
  - HAL_HASH_MD5_Accumulate()
- SHA1
  - HAL_HASH_SHA1_Start()
  - HAL_HASH_SHA1_Accumulate()

For a single buffer to be hashed, user can resort to HAL_HASH_xxx_Start().

In case of multi-buffer HASH processing (a single digest is computed while several buffers are fed to the IP), the user can resort to successive calls to HAL_HASH_xxx_Accumulate() and wrap-up the digest computation by a call to HAL_HASH_xxx_Start().

This section contains the following APIs:

- ***HAL_HASH_MD5_Start()***
- ***HAL_HASH_MD5_Accumulate()***
- ***HAL_HASH_SHA1_Start()***
- ***HAL_HASH_SHA1_Accumulate()***

30.2.4 Interruption mode HASH processing functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- MD5
  - HAL_HASH_MD5_Start_IT()
- SHA1
  - HAL_HASH_SHA1_Start_IT()

API HAL_HASH_IRQHandler() manages each HASH interruption.

Note that HAL_HASH_IRQHandler() manages as well HASH IP interruptions when in HMAC processing mode.

This section contains the following APIs:

- ***HAL_HASH_MD5_Start_IT()***
- ***HAL_HASH_SHA1_Start_IT()***
- ***HAL_HASH_IRQHandler()***

30.2.5 DMA mode HASH processing functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- MD5
  - HAL_HASH_MD5_Start_DMA()
  - HAL_HASH_MD5_Finish()
- SHA1
  - HAL_HASH_SHA1_Start_DMA()
  - HAL_HASH_SHA1_Finish()

When resorting to DMA mode to enter the data in the IP, user must resort to HAL_HASH_xxx_Start_DMA() then read the resulting digest with HAL_HASH_xxx_Finish().

In case of multi-buffer HASH processing, MDMAT bit must first be set before the successive calls to HAL_HASH_xxx_Start_DMA(). Then, MDMAT bit needs to be reset before the last call to HAL_HASH_xxx_Start_DMA(). Digest is finally retrieved thanks to HAL_HASH_xxx_Finish().

This section contains the following APIs:

- ***HAL_HASH_MD5_Start_DMA()***
- ***HAL_HASH_MD5_Finish()***
- ***HAL_HASH_SHA1_Start_DMA()***
- ***HAL_HASH_SHA1_Finish()***

30.2.6 Polling mode HMAC processing functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- MD5
  - HAL_HMAC_MD5_Start()
- SHA1
  - HAL_HMAC_SHA1_Start()

This section contains the following APIs:

- ***HAL_HMAC_MD5_Start()***
- ***HAL_HMAC_SHA1_Start()***

30.2.7 Interrupt mode HMAC processing functions

This section provides functions allowing to calculate in interrupt mode the HMAC value using one of the following algorithms:

- MD5
  - HAL_HMAC_MD5_Start_IT()
- SHA1
  - HAL_HMAC_SHA1_Start_IT()

This section contains the following APIs:

- ***HAL_HMAC_MD5_Start_IT()***

- ***HAL_HMAC_SHA1_Start_IT()***

30.2.8 DMA mode HMAC processing functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- MD5
  - HAL_HMAC_MD5_Start_DMA()
- SHA1
  - HAL_HMAC_SHA1_Start_DMA()

When resorting to DMA mode to enter the data in the IP for HMAC processing, user must resort to HAL_HMAC_xxx_Start_DMA() then read the resulting digest with HAL_HASH_xxx_Finish().

This section contains the following APIs:

- ***HAL_HMAC_MD5_Start_DMA()***
- ***HAL_HMAC_SHA1_Start_DMA()***

30.2.9 Peripheral State methods

This section permits to get in run-time the state and the peripheral handle status of the peripheral:

- HAL_HASH_GetState()
- HAL_HASH_GetStatus()

Additionally, this subsection provides functions allowing to save and restore the HASH or HMAC processing context in case of calculation suspension:

- HAL_HASH_ContextSaving()
- HAL_HASH_ContextRestoring()

This subsection provides functions allowing to suspend the HASH processing

- when input are fed to the IP by software
  - HAL_HASH_SwFeed_ProcessSuspend()
- when input are fed to the IP by DMA
  - HAL_HASH_DMAFeed_ProcessSuspend()

This section contains the following APIs:

- ***HAL_HASH_GetState()***
- ***HAL_HASH_GetStatus()***
- ***HAL_HASH_ContextSaving()***
- ***HAL_HASH_ContextRestoring()***
- ***HAL_HASH_SwFeed_ProcessSuspend()***
- ***HAL_HASH_DMAFeed_ProcessSuspend()***

30.2.10 Detailed description of functions

HAL_HASH_Init

Function name	**HAL_StatusTypeDef HAL_HASH_Init (HASH_HandleTypeDef * hhash)**
Function description	Initialize the HASH according to the specified parameters in the HASH_HandleTypeDef and create the associated handle.

Parameters	• **hhash:** HASH handle
Return values	• **HAL:** status
Notes	• Only MDMAT and DATATYPE bits of HASH IP are set by HAL_HASH_Init(), other configuration bits are set by HASH or HMAC processing APIs. • MDMAT bit is systematically reset by HAL_HASH_Init(). To set it for multi-buffer HASH processing, user needs to resort to __HAL_HASH_SET_MDMAT() macro. For HMAC multi-buffer processing, the relevant APIs manage themselves the MDMAT bit.

HAL_HASH_DeInit

Function name	**HAL_StatusTypeDef HAL_HASH_DeInit (HASH_HandleTypeDef * hhash)**
Function description	DeInitialize the HASH peripheral.
Parameters	• **hhash:** HASH handle.
Return values	• **HAL:** status

HAL_HASH_MspInit

Function name	**void HAL_HASH_MspInit (HASH_HandleTypeDef * hhash)**
Function description	Initialize the HASH MSP.
Parameters	• **hhash:** HASH handle.
Return values	• **None:**

HAL_HASH_MspDeInit

Function name	**void HAL_HASH_MspDeInit (HASH_HandleTypeDef * hhash)**
Function description	DeInitialize the HASH MSP.
Parameters	• **hhash:** HASH handle.
Return values	• **None:**

HAL_HASH_InCpltCallback

Function name	**void HAL_HASH_InCpltCallback (HASH_HandleTypeDef * hhash)**
Function description	Input data transfer complete call back.
Parameters	• **hhash:** HASH handle.
Return values	• **None:**
Notes	• HAL_HASH_InCpltCallback() is called when the complete input message has been fed to the IP. This API is invoked only when input data are entered under interruption or thru DMA. • In case of HASH or HMAC multi-buffer DMA feeding case (MDMAT bit set), HAL_HASH_InCpltCallback() is called at the

end of each buffer feeding to the IP.

HAL_HASH_DgstCpltCallback

Function name	**void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)**
Function description	Digest computation complete call back.
Parameters	• **hhash:** HASH handle.
Return values	• **None:**
Notes	• HAL_HASH_DgstCpltCallback() is used under interruption, is not relevant with DMA.

HAL_HASH_ErrorCallback

Function name	**void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)**
Function description	Error callback.
Parameters	• **hhash:** HASH handle.
Return values	• **None:**
Notes	• Code user can resort to hhash->Status (HAL_ERROR, HAL_TIMEOUT,...) to retrieve the error type.

HAL_HASH_SHA1_Start

Function name	**HAL_StatusTypeDef HAL_HASH_SHA1_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Initialize the HASH peripheral in SHA1 mode, next process pInBuffer then read the computed digest.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes. • **Timeout:** Timeout value
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer.

HAL_HASH_MD5_Start

Function name	**HAL_StatusTypeDef HAL_HASH_MD5_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Initialize the HASH peripheral in MD5 mode, next process pInBuffer then read the computed digest.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed).

- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value

Return values
- **HAL:** status

Notes
- Digest is available in pOutBuffer.

HAL_HASH_MD5_Accumulate

Function name	**HAL_StatusTypeDef HAL_HASH_MD5_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	If not already done, initialize the HASH peripheral in MD5 mode then processes pInBuffer.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes, must be a multiple of 4.
Return values	• **HAL:** status
Notes	• Consecutive calls to HAL_HASH_MD5_Accumulate() can be used to feed several input buffers back-to-back to the IP that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_MD5_Start(). • Field hhash->Phase of HASH handle is tested to check whether or not the IP has already been initialized. • Digest is not retrieved by this API, user must resort to HAL_HASH_MD5_Start() to read it, feeding at the same time the last input buffer to the IP. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASH_MD5_Start() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASH_SHA1_Accumulate

Function name	**HAL_StatusTypeDef HAL_HASH_SHA1_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	If not already done, initialize the HASH peripheral in SHA1 mode then processes pInBuffer.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes, must be a multiple of 4.
Return values	• **HAL:** status
Notes	• Consecutive calls to HAL_HASH_SHA1_Accumulate() can be used to feed several input buffers back-to-back to the IP that will yield a single HASH signature once all buffers have been

entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_SHA1_Start().
- Field hhash->Phase of HASH handle is tested to check whether or not the IP has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASH_SHA1_Start() to read it, feeding at the same time the last input buffer to the IP.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASH_SHA1_Start() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASH_SHA1_Start_IT

Function name	**HAL_StatusTypeDef HAL_HASH_SHA1_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)**
Function description	Initialize the HASH peripheral in SHA1 mode, next process pInBuffer then read the computed digest in interruption mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer.

HAL_HASH_MD5_Start_IT

Function name	**HAL_StatusTypeDef HAL_HASH_MD5_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)**
Function description	Initialize the HASH peripheral in MD5 mode, next process pInBuffer then read the computed digest in interruption mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer.

HAL_HASH_IRQHandler

Function name	**void HAL_HASH_IRQHandler (HASH_HandleTypeDef * hhash)**
Function description	Handle HASH interrupt request.
Parameters	• **hhash:** HASH handle.
Return values	• **None:**

Notes	• HAL_HASH_IRQHandler() handles interrupts in HMAC processing as well. • In case of error reported during the HASH interruption processing, HAL_HASH_ErrorCallback() API is called so that user code can manage the error. The error type is available in hhash->Status field.

HAL_HASH_SHA1_Start_DMA

Function name	**HAL_StatusTypeDef HAL_HASH_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	Initialize the HASH peripheral in SHA1 mode then initiate a DMA transfer to feed the input buffer to the IP.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Once the DMA transfer is finished, HAL_HASH_SHA1_Finish() API must be called to retrieve the computed digest.

HAL_HASH_SHA1_Finish

Function name	**HAL_StatusTypeDef HAL_HASH_SHA1_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Return the computed digest in SHA1 mode.
Parameters	• **hhash:** HASH handle. • **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes. • **Timeout:** Timeout value.
Return values	• **HAL:** status
Notes	• The API waits for DCIS to be set then reads the computed digest. • HAL_HASH_SHA1_Finish() can be used as well to retrieve the digest in HMAC SHA1 mode.

HAL_HASH_MD5_Start_DMA

Function name	**HAL_StatusTypeDef HAL_HASH_MD5_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	Initialize the HASH peripheral in MD5 mode then initiate a DMA transfer to feed the input buffer to the IP.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes.

Return values	• **HAL:** status
Notes	• Once the DMA transfer is finished, HAL_HASH_MD5_Finish() API must be called to retrieve the computed digest.

HAL_HASH_MD5_Finish

Function name	**HAL_StatusTypeDef HAL_HASH_MD5_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Return the computed digest in MD5 mode.
Parameters	• **hhash:** HASH handle. • **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes. • **Timeout:** Timeout value.
Return values	• **HAL:** status
Notes	• The API waits for DCIS to be set then reads the computed digest. • HAL_HASH_MD5_Finish() can be used as well to retrieve the digest in HMAC MD5 mode.

HAL_HMAC_SHA1_Start

Function name	**HAL_StatusTypeDef HAL_HMAC_SHA1_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Initialize the HASH peripheral in HMAC SHA1 mode, next process pInBuffer then read the computed digest.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes. • **Timeout:** Timeout value.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_MD5_Start

Function name	**HAL_StatusTypeDef HAL_HMAC_MD5_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Initialize the HASH peripheral in HMAC MD5 mode, next process pInBuffer then read the computed digest.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed).

- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value.

Return values
- **HAL:** status

Notes
- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_MD5_Start_IT

Function name
HAL_StatusTypeDef HAL_HMAC_MD5_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description
Initialize the HASH peripheral in HMAC MD5 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters
- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.

Return values
- **HAL:** status

Notes
- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_SHA1_Start_IT

Function name
HAL_StatusTypeDef HAL_HMAC_SHA1_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description
Initialize the HASH peripheral in HMAC SHA1 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters
- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.

Return values
- **HAL:** status

Notes
- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_SHA1_Start_DMA

Function name	**HAL_StatusTypeDef HAL_HMAC_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	Initialize the HASH peripheral in HMAC SHA1 mode then initiate the required DMA transfers to feed the key and the input buffer to the IP.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASH_SHA1_Finish() API must be called to retrieve the computed digest. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HMAC_MD5_Start_DMA

Function name	**HAL_StatusTypeDef HAL_HMAC_MD5_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	Initialize the HASH peripheral in HMAC MD5 mode then initiate the required DMA transfers to feed the key and the input buffer to the IP.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASH_MD5_Finish() API must be called to retrieve the computed digest. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes)

doesn't have to be a multiple of 4.

HAL_HASH_GetState

Function name	**HAL_HASH_StateTypeDef HAL_HASH_GetState (HASH_HandleTypeDef * hhash)**
Function description	Return the HASH handle state.
Parameters	• **hhash:** HASH handle.
Return values	• **HAL:** HASH state
Notes	• The API yields the current state of the handle (BUSY, READY,...).

HAL_HASH_GetStatus

Function name	**HAL_StatusTypeDef HAL_HASH_GetStatus (HASH_HandleTypeDef * hhash)**
Function description	Return the HASH HAL status.
Parameters	• **hhash:** HASH handle.
Return values	• **HAL:** status
Notes	• The API yields the HAL status of the handle: it is the result of the latest HASH processing and allows to report any issue (e.g. HAL_TIMEOUT).

HAL_HASH_ContextSaving

Function name	**void HAL_HASH_ContextSaving (HASH_HandleTypeDef * hhash, uint8_t * pMemBuffer)**
Function description	Save the HASH context in case of processing suspension.
Parameters	• **hhash:** HASH handle. • **pMemBuffer:** pointer to the memory buffer where the HASH context is saved.
Return values	• **None:**
Notes	• The IMR, STR, CR then all the CSR registers are saved in that order. Only the r/w bits are read to be restored later on. • By default, all the context swap registers (there are HASH_NUMBER_OF_CSR_REGISTERS of those) are saved. • pMemBuffer points to a buffer allocated by the user. The buffer size must be at least (HASH_NUMBER_OF_CSR_REGISTERS + 3) * 4 uint8 long.

HAL_HASH_ContextRestoring

Function name	**void HAL_HASH_ContextRestoring (HASH_HandleTypeDef * hhash, uint8_t * pMemBuffer)**
Function description	Restore the HASH context in case of processing resumption.

Parameters	• **hhash:** HASH handle. • **pMemBuffer:** pointer to the memory buffer where the HASH context is stored.
Return values	• **None:**
Notes	• The IMR, STR, CR then all the CSR registers are restored in that order. Only the r/w bits are restored. • By default, all the context swap registers (HASH_NUMBER_OF_CSR_REGISTERS of those) are restored (all of them have been saved by default beforehand).

HAL_HASH_SwFeed_ProcessSuspend

Function name	**void HAL_HASH_SwFeed_ProcessSuspend (HASH_HandleTypeDef * hhash)**
Function description	Initiate HASH processing suspension when in polling or interruption mode.
Parameters	• **hhash:** HASH handle.
Return values	• **None:**
Notes	• Set the handle field SuspendRequest to the appropriate value so that the on-going HASH processing is suspended as soon as the required conditions are met. Note that the actual suspension is carried out by the functions HASH_WriteData() in polling mode and HASH_IT() in interruption mode.

HAL_HASH_DMAFeed_ProcessSuspend

Function name	**HAL_StatusTypeDef HAL_HASH_DMAFeed_ProcessSuspend (HASH_HandleTypeDef * hhash)**
Function description	Suspend the HASH processing when in DMA mode.
Parameters	• **hhash:** HASH handle.
Return values	• **HAL:** status
Notes	• When suspension attempt occurs at the very end of a DMA transfer and all the data have already been entered in the IP, hhash->State is set to HAL_HASH_STATE_READY and the API returns HAL_ERROR. It is recommended to wrap-up the processing in reading the digest as usual.

HASH_Start

Function name	**HAL_StatusTypeDef HASH_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout, uint32_t Algorithm)**
Function description	Initialize the HASH peripheral, next process pInBuffer then read the computed digest.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest.

	• **Timeout:** Timeout value. • **Algorithm:** HASH algorithm.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer.

HASH_Accumulate

Function name	**HAL_StatusTypeDef HASH_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)**
Function description	If not already done, initialize the HASH peripheral then processes pInBuffer.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes, must be a multiple of 4. • **Algorithm:** HASH algorithm.
Return values	• **HAL:** status
Notes	• Field hhash->Phase of HASH handle is tested to check whether or not the IP has already been initialized. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HASH_Start_IT

Function name	**HAL_StatusTypeDef HASH_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Algorithm)**
Function description	Initialize the HASH peripheral, next process pInBuffer then read the computed digest in interruption mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. • **Algorithm:** HASH algorithm.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer.

HASH_Start_DMA

Function name	**HAL_StatusTypeDef HASH_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)**
Function description	Initialize the HASH peripheral then initiate a DMA transfer to feed the input buffer to the IP.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed).

	• **Size:** length of the input buffer in bytes. • **Algorithm:** HASH algorithm.
Return values	• **HAL:** status
Notes	• If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HASH_Finish

Function name	**HAL_StatusTypeDef HASH_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Return the computed digest.
Parameters	• **hhash:** HASH handle. • **pOutBuffer:** pointer to the computed digest. • **Timeout:** Timeout value.
Return values	• **HAL:** status
Notes	• The API waits for DCIS to be set then reads the computed digest.

HMAC_Start

Function name	**HAL_StatusTypeDef HMAC_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout, uint32_t Algorithm)**
Function description	Initialize the HASH peripheral in HMAC mode, next process pInBuffer then read the computed digest.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. • **Timeout:** Timeout value. • **Algorithm:** HASH algorithm.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HMAC_Start_IT

Function name	**HAL_StatusTypeDef HMAC_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Algorithm)**
Function description	Initialize the HASH peripheral in HMAC mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. • **Algorithm:** HASH algorithm.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HMAC_Start_DMA

Function name	**HAL_StatusTypeDef HMAC_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)**
Function description	Initialize the HASH peripheral in HMAC mode then initiate the required DMA transfers to feed the key and the input buffer to the IP.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **Algorithm:** HASH algorithm.
Return values	• **HAL:** status
Notes	• Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • In case of multi-buffer HMAC processing, the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only the length of the last buffer of the thread doesn't have to be a multiple of 4.

30.3 HASH Firmware driver defines

30.3.1 HASH

HASH algorithm mode

HASH_ALGOMODE_HASH	Algorithm is HASH
HASH_ALGOMODE_HMAC	Algorithm is HMAC

HASH algorithm selection

HASH_ALGOSELECTION_SHA1	HASH function is SHA1
HASH_ALGOSELECTION_SHA224	HASH function is SHA224
HASH_ALGOSELECTION_SHA256	HASH function is SHA256
HASH_ALGOSELECTION_MD5	HASH function is MD5

HASH API alias

HAL_HASHEx_IRQHandler	is re-directed to

HASH input data type

HASH_DATATYPE_32B	32-bit data. No swapping
HASH_DATATYPE_16B	16-bit data. Each half word is swapped
HASH_DATATYPE_8B	8-bit data. All bytes are swapped
HASH_DATATYPE_1B	1-bit data. In the word all bits are swapped

HASH Digest Calculation Status

HASH_DIGEST_CALCULATION_NOT_STARTED	DCAL not set after input data written in DIN register
HASH_DIGEST_CALCULATION_STARTED	DCAL set after input data written in DIN register

HASH DMA suspension words limit

HASH_DMA_SUSPENSION_WORDS_LIMIT	Number of words below which DMA suspension is aborted

HASH Exported Macros

__HAL_HASH_GET_FLAG

Description:

- Check whether or not the specified HASH flag is set.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - HASH_FLAG_DINIS A new block can be entered into the input buffer.
  - HASH_FLAG_DCIS Digest calculation complete.
  - HASH_FLAG_DMAS DMA interface is enabled (DMAE=1) or a transfer is ongoing.
  - HASH_FLAG_BUSY The hash core is Busy: processing a block of data.
  - HASH_FLAG_DINNE DIN not empty: the input buffer contains at least one word of data.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_HASH_CLEAR_FLAG

Description:

- Clear the specified HASH flag.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:

- HASH_FLAG_DINIS A new block can be entered into the input buffer.
- HASH_FLAG_DCIS Digest calculation complete

Return value:

- None

__HAL_HASH_ENABLE_IT

Description:

- Enable the specified HASH interrupt.

Parameters:

- __INTERRUPT__: specifies the HASH interrupt source to enable. This parameter can be one of the following values:
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:

- None

__HAL_HASH_DISABLE_IT

Description:

- Disable the specified HASH interrupt.

Parameters:

- __INTERRUPT__: specifies the HASH interrupt source to disable. This parameter can be one of the following values:
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:

- None

__HAL_HASH_RESET_HANDLE_STATE

Description:

- Reset HASH handle state.

Parameters:

- __HANDLE__: HASH handle.

Return value:

- None

__HAL_HASH_RESET_HANDLE_STATUS

Description:

- Reset HASH handle status.

Parameters:

- __HANDLE__: HASH handle.

Return value:

- None

__HAL_HASH_SET_MDMAT

Description:

- Enable the multi-buffer DMA transfer mode.

Return value:

- None

Notes:

- This bit is set when hashing large files when multiple DMA transfers are needed.

__HAL_HASH_RESET_MDMAT

Description:

- Disable the multi-buffer DMA transfer mode.

Return value:

- None

__HAL_HASH_START_DIGEST

Description:

- Start the digest computation.

Return value:

- None

__HAL_HASH_SET_NBVALIDBITS

Description:

- Set the number of valid bits in the last word written in data register DIN.

Parameters:

- __SIZE__: size in bytes of last data written in Data register.

Return value:

- None

__HAL_HASH_INIT

Description:

- Reset the HASH core.

Return value:

- None

HASH flags definitions

HASH_FLAG_DINIS 16 locations are free in the DIN: a new block can be entered in the IP

HASH_FLAG_DCIS	Digest calculation complete
HASH_FLAG_DMAS	DMA interface is enabled (DMAE=1) or a transfer is ongoing
HASH_FLAG_BUSY	The hash core is Busy, processing a block of data
HASH_FLAG_DINNE	DIN not empty: the input buffer contains at least one word of data

HMAC key length type

HASH_HMAC_KEYTYPE_SHORTKEY	HMAC Key size is <= 64 bytes
HASH_HMAC_KEYTYPE_LONGKEY	HMAC Key size is> 64 bytes

HASH interrupts definitions

HASH_IT_DINI	A new block can be entered into the input buffer (DIN)
HASH_IT_DCI	Digest calculation complete

HASH Number of Context Swap Registers

HASH_NUMBER_OF_CSR_REGISTERS	Number of Context Swap Registers

HASH TimeOut Value

HASH_TIMEOUTVALUE	Time-out value

31 HAL HASH Extension Driver

31.1 HASHEx Firmware driver API description

31.1.1 HASH peripheral extended features

The SHA-224 and SHA-256 HASH and HMAC processing can be carried out exactly the same way as for SHA-1 or MD-5 algorithms.

1. Three modes are available.
   a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished, e.g. HAL_HASHEx_xxx_Start()
   b. Interrupt mode: processing APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_HASHEx_xxx_Start_IT()
   c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA, e.g. HAL_HASHEx_xxx_Start_DMA(). Note that in DMA mode, a call to HAL_HASHEx_xxx_Finish() is then required to retrieve the digest.
2. Multi-buffer processing is possible in polling and DMA mode.
   a. In polling mode, only multi-buffer HASH processing is possible. API HAL_HASHEx_xxx_Accumulate() must be called for each input buffer, except for the last one. User must resort to HAL_HASHEx_xxx_Start() to enter the last one and retrieve as well the computed digest.
   b. In DMA mode, multi-buffer HASH and HMAC processing are possible.
      – HASH processing: once initialization is done, MDMAT bit must be set thru __HAL_HASH_SET_MDMAT() macro. From that point, each buffer can be fed to the IP thru HAL_HASHEx_xxx_Start_DMA() API. Before entering the last buffer, reset the MDMAT bit with __HAL_HASH_RESET_MDMAT() macro then wrap-up the HASH processing in feeding the last input buffer thru the same API HAL_HASHEx_xxx_Start_DMA(). The digest can then be retrieved with a call to API HAL_HASHEx_xxx_Finish().
      – HMAC processing (MD-5, SHA-1, SHA-224 and SHA-256 must all resort to extended functions): after initialization, the key and the first input buffer are entered in the IP with the API HAL_HMACEx_xxx_Step1_2_DMA(). This carries out HMAC step 1 and starts step 2. The following buffers are next entered with the API HAL_HMACEx_xxx_Step2_DMA(). At this point, the HMAC processing is still carrying out step 2. Then, step 2 for the last input buffer and step 3 are carried out by a single call to HAL_HMACEx_xxx_Step2_3_DMA(). The digest can finally be retrieved with a call to API HAL_HASH_xxx_Finish() for MD-5 and SHA-1, to HAL_HASHEx_xxx_Finish() for SHA-224 and SHA-256.

31.1.2 Polling mode HASH extended processing functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- SHA224
  – HAL_HASHEx_SHA224_Start()
  – HAL_HASHEx_SHA224_Accumulate()
- SHA256
  – HAL_HASHEx_SHA256_Start()
  – HAL_HASHEx_SHA256_Accumulate()

For a single buffer to be hashed, user can resort to HAL_HASH_xxx_Start().

In case of multi-buffer HASH processing (a single digest is computed while several buffers are fed to the IP), the user can resort to successive calls to HAL_HASHEx_xxx_Accumulate() and wrap-up the digest computation by a call to HAL_HASHEx_xxx_Start().

This section contains the following APIs:

- ***HAL_HASHEx_SHA224_Start()***
- ***HAL_HASHEx_SHA224_Accumulate()***
- ***HAL_HASHEx_SHA256_Start()***
- ***HAL_HASHEx_SHA256_Accumulate()***

31.1.3 Interruption mode HASH extended processing functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- SHA224
  - HAL_HASHEx_SHA224_Start_IT()
- SHA256
  - HAL_HASHEx_SHA256_Start_IT()

This section contains the following APIs:

- ***HAL_HASHEx_SHA224_Start_IT()***
- ***HAL_HASHEx_SHA256_Start_IT()***

31.1.4 DMA mode HASH extended processing functionss

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- SHA224
  - HAL_HASHEx_SHA224_Start_DMA()
  - HAL_HASHEx_SHA224_Finish()
- SHA256
  - HAL_HASHEx_SHA256_Start_DMA()
  - HAL_HASHEx_SHA256_Finish()

When resorting to DMA mode to enter the data in the IP, user must resort to HAL_HASHEx_xxx_Start_DMA() then read the resulting digest with HAL_HASHEx_xxx_Finish().

In case of multi-buffer HASH processing, MDMAT bit must first be set before the successive calls to HAL_HASHEx_xxx_Start_DMA(). Then, MDMAT bit needs to be reset before the last call to HAL_HASHEx_xxx_Start_DMA(). Digest is finally retrieved thanks to HAL_HASHEx_xxx_Finish().

This section contains the following APIs:

- ***HAL_HASHEx_SHA224_Start_DMA()***
- ***HAL_HASHEx_SHA224_Finish()***
- ***HAL_HASHEx_SHA256_Start_DMA()***
- ***HAL_HASHEx_SHA256_Finish()***

31.1.5 Polling mode HMAC extended processing functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL_HMACEx_SHA224_Start()
- SHA256
  - HAL_HMACEx_SHA256_Start()

This section contains the following APIs:

- ***HAL_HMACEx_SHA224_Start()***
- ***HAL_HMACEx_SHA256_Start()***

31.1.6 Interrupt mode HMAC extended processing functions

This section provides functions allowing to calculate in interrupt mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL_HMACEx_SHA224_Start_IT()
- SHA256
  - HAL_HMACEx_SHA256_Start_IT()

This section contains the following APIs:

- ***HAL_HMACEx_SHA224_Start_IT()***
- ***HAL_HMACEx_SHA256_Start_IT()***

31.1.7 DMA mode HMAC extended processing functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL_HMACEx_SHA224_Start_DMA()
- SHA256
  - HAL_HMACEx_SHA256_Start_DMA()

When resorting to DMA mode to enter the data in the IP for HMAC processing, user must resort to HAL_HMACEx_xxx_Start_DMA() then read the resulting digest with HAL_HASHEx_xxx_Finish().

This section contains the following APIs:

- ***HAL_HMACEx_SHA224_Start_DMA()***
- ***HAL_HMACEx_SHA256_Start_DMA()***

31.1.8 Multi-buffer DMA mode HMAC extended processing functions

This section provides functions to manage HMAC multi-buffer DMA-based processing for MD5, SHA1, SHA224 and SHA256 algorithms.

- MD5
  - HAL_HMACEx_MD5_Step1_2_DMA()
  - HAL_HMACEx_MD5_Step2_DMA()
  - HAL_HMACEx_MD5_Step2_3_DMA()
- SHA1
  - HAL_HMACEx_SHA1_Step1_2_DMA()
  - HAL_HMACEx_SHA1_Step2_DMA()
  - HAL_HMACEx_SHA1_Step2_3_DMA()
- SHA256
  - HAL_HMACEx_SHA224_Step1_2_DMA()
  - HAL_HMACEx_SHA224_Step2_DMA()

   - HAL_HMACEx_SHA224_Step2_3_DMA()
- SHA256
   - HAL_HMACEx_SHA256_Step1_2_DMA()
   - HAL_HMACEx_SHA256_Step2_DMA()
   - HAL_HMACEx_SHA256_Step2_3_DMA()

User must first start-up the multi-buffer DMA-based HMAC computation in calling HAL_HMACEx_xxx_Step1_2_DMA(). This carries out HMAC step 1 and intiates step 2 with the first input buffer.

The following buffers are next fed to the IP with a call to the API HAL_HMACEx_xxx_Step2_DMA(). There may be several consecutive calls to this API.

Multi-buffer DMA-based HMAC computation is wrapped up by a call to HAL_HMACEx_xxx_Step2_3_DMA(). This finishes step 2 in feeding the last input buffer to the IP then carries out step 3.

Digest is retrieved by a call to HAL_HASH_xxx_Finish() for MD-5 or SHA-1, to HAL_HASHEx_xxx_Finish() for SHA-224 or SHA-256.

If only two buffers need to be consecutively processed, a call to HAL_HMACEx_xxx_Step1_2_DMA() followed by a call to HAL_HMACEx_xxx_Step2_3_DMA() is sufficient.

This section contains the following APIs:

- ***HAL_HMACEx_MD5_Step1_2_DMA()***
- ***HAL_HMACEx_MD5_Step2_DMA()***
- ***HAL_HMACEx_MD5_Step2_3_DMA()***
- ***HAL_HMACEx_SHA1_Step1_2_DMA()***
- ***HAL_HMACEx_SHA1_Step2_DMA()***
- ***HAL_HMACEx_SHA1_Step2_3_DMA()***
- ***HAL_HMACEx_SHA224_Step1_2_DMA()***
- ***HAL_HMACEx_SHA224_Step2_DMA()***
- ***HAL_HMACEx_SHA224_Step2_3_DMA()***
- ***HAL_HMACEx_SHA256_Step1_2_DMA()***
- ***HAL_HMACEx_SHA256_Step2_DMA()***
- ***HAL_HMACEx_SHA256_Step2_3_DMA()***

31.1.9 Detailed description of functions

HAL_HASHEx_SHA224_Start

Function name	**HAL_StatusTypeDef HAL_HASHEx_SHA224_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Initialize the HASH peripheral in SHA224 mode, next process pInBuffer then read the computed digest.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes. • **Timeout:** Timeout value
Return values	• **HAL:** status

Notes	• Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Accumulate

Function name	**HAL_StatusTypeDef HAL_HASHEx_SHA224_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	If not already done, initialize the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes, must be a multiple of 4.
Return values	• **HAL:** status
Notes	• Consecutive calls to HAL_HASHEx_SHA224_Accumulate() can be used to feed several input buffers back-to-back to the IP that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA224_Start(). • Field hhash->Phase of HASH handle is tested to check whether or not the IP has already been initialized. • Digest is not retrieved by this API, user must resort to HAL_HASHEx_SHA224_Start() to read it, feeding at the same time the last input buffer to the IP. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA224_Start() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA256_Start

Function name	**HAL_StatusTypeDef HAL_HASHEx_SHA256_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Initialize the HASH peripheral in SHA256 mode, next process pInBuffer then read the computed digest.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes. • **Timeout:** Timeout value
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Accumulate

Function name	**HAL_StatusTypeDef HAL_HASHEx_SHA256_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t**

	Size)
Function description	If not already done, initialize the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes, must be a multiple of 4.
Return values	• **HAL:** status
Notes	• Consecutive calls to HAL_HASHEx_SHA256_Accumulate() can be used to feed several input buffers back-to-back to the IP that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA256_Start(). • Field hhash->Phase of HASH handle is tested to check whether or not the IP has already been initialized. • Digest is not retrieved by this API, user must resort to HAL_HASHEx_SHA256_Start() to read it, feeding at the same time the last input buffer to the IP. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA256_Start() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA224_Start_IT

Function name	**HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)**
Function description	Initialize the HASH peripheral in SHA224 mode, next process pInBuffer then read the computed digest in interruption mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Start_IT

Function name	**HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)**
Function description	Initialize the HASH peripheral in SHA256 mode, next process pInBuffer then read the computed digest in interruption mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes.

	• **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Start_DMA

Function name	**HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	Initialize the HASH peripheral in SHA224 mode then initiate a DMA transfer to feed the input buffer to the IP.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Once the DMA transfer is finished, HAL_HASHEx_SHA224_Finish() API must be called to retrieve the computed digest.

HAL_HASHEx_SHA224_Finish

Function name	**HAL_StatusTypeDef HAL_HASHEx_SHA224_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Return the computed digest in SHA224 mode.
Parameters	• **hhash:** HASH handle. • **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes. • **Timeout:** Timeout value.
Return values	• **HAL:** status
Notes	• The API waits for DCIS to be set then reads the computed digest. • HAL_HASHEx_SHA224_Finish() can be used as well to retrieve the digest in HMAC SHA224 mode.

HAL_HASHEx_SHA256_Start_DMA

Function name	**HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	Initialize the HASH peripheral in SHA256 mode then initiate a DMA transfer to feed the input buffer to the IP.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status

Notes	• Once the DMA transfer is finished, HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HASHEx_SHA256_Finish

Function name	**HAL_StatusTypeDef HAL_HASHEx_SHA256_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Return the computed digest in SHA256 mode.
Parameters	• **hhash:** HASH handle. • **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes. • **Timeout:** Timeout value.
Return values	• **HAL:** status
Notes	• The API waits for DCIS to be set then reads the computed digest. • HAL_HASHEx_SHA256_Finish() can be used as well to retrieve the digest in HMAC SHA256 mode.

HAL_HMACEx_SHA224_Start

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA224_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Initialize the HASH peripheral in HMAC SHA224 mode, next process pInBuffer then read the computed digest.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes. • **Timeout:** Timeout value.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA256_Start

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA256_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)**
Function description	Initialize the HASH peripheral in HMAC SHA256 mode, next process pInBuffer then read the computed digest.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes.

	• **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes. • **Timeout:** Timeout value.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA224_Start_IT

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)**
Function description	Initialize the HASH peripheral in HMAC SHA224 mode, next process pInBuffer then read the computed digest in interrupt mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA256_Start_IT

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA256_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)**
Function description	Initialize the HASH peripheral in HMAC SHA256 mode, next process pInBuffer then read the computed digest in interrupt mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes. • **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
Return values	• **HAL:** status
Notes	• Digest is available in pOutBuffer. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA224_Start_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	Initialize the HASH peripheral in HMAC SHA224 mode then initiate the required DMA transfers to feed the key and the input buffer to the IP.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA224_Finish() API must be called to retrieve the computed digest. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HMACEx_SHA256_Start_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	Initialize the HASH peripheral in HMAC SHA224 mode then initiate the required DMA transfers to feed the key and the input buffer to the IP.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (buffer to be hashed). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes)

doesn't have to be a multiple of 4.

HAL_HMACEx_MD5_Step1_2_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_MD5_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	MD5 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 1 consists in writing the inner hash function key in the IP, step 2 consists in writing the message text. • The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the IP. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_MD5_Step2_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_MD5_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	MD5 HMAC step 2 in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 2 consists in writing the message text in the IP. • The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_MD5_Step2_3_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_MD5_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t**

	Size)
Function description	MD5 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 2 consists in writing the message text in the IP, step 3 consists in writing the outer hash function key. • The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA1_Step1_2_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA1_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	SHA1 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 1 consists in writing the inner hash function key in the IP, step 2 consists in writing the message text. • The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the IP. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA1_Step2_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA1_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	SHA1 HMAC step 2 in multi-buffer DMA mode.

Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 2 consists in writing the message text in the IP. • The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA1_Step2_3_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA1_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	SHA1 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 2 consists in writing the message text in the IP, step 3 consists in writing the outer hash function key. • The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA224_Step1_2_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA224_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	SHA224 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.

Return values	• **HAL:** status
Notes	• Step 1 consists in writing the inner hash function key in the IP, step 2 consists in writing the message text. • The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the IP. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA224_Step2_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA224_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	SHA224 HMAC step 2 in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 2 consists in writing the message text in the IP. • The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA224_Step2_3_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA224_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	SHA224 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 2 consists in writing the message text in the IP, step 3 consists in writing the outer hash function key. • The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last

- one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA256_Step1_2_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA256_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	SHA256 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 1 consists in writing the inner hash function key in the IP, step 2 consists in writing the message text. • The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the IP. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA256_Step2_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA256_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	SHA256 HMAC step 2 in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 2 consists in writing the message text in the IP. • The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA256_Step2_3_DMA

Function name	**HAL_StatusTypeDef HAL_HMACEx_SHA256_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)**
Function description	SHA256 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.
Parameters	• **hhash:** HASH handle. • **pInBuffer:** pointer to the input buffer (message buffer). • **Size:** length of the input buffer in bytes.
Return values	• **HAL:** status
Notes	• Step 2 consists in writing the message text in the IP, step 3 consists in writing the outer hash function key. • The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3. • Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize. • Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

32 HAL HCD Generic Driver

32.1 HCD Firmware driver registers structures

32.1.1 HCD_HandleTypeDef

Data Fields

- ***HCD_TypeDef * Instance***
- ***HCD_InitTypeDef Init***
- ***HCD_HCTypeDef hc***
- ***HAL_LockTypeDef Lock***
- ***__IO HCD_StateTypeDef State***
- ***void * pData***

Field Documentation

- ***HCD_TypeDef* HCD_HandleTypeDef::Instance***
  Register base address
- ***HCD_InitTypeDef HCD_HandleTypeDef::Init***
  HCD required parameters
- ***HCD_HCTypeDef HCD_HandleTypeDef::hc[15]***
  Host channels parameters
- ***HAL_LockTypeDef HCD_HandleTypeDef::Lock***
  HCD peripheral status
- ***__IO HCD_StateTypeDef HCD_HandleTypeDef::State***
  HCD communication state
- ***void* HCD_HandleTypeDef::pData***
  Pointer Stack Handler

32.2 HCD Firmware driver API description

32.2.1 How to use this driver

1. Declare a HCD_HandleTypeDef handle structure, for example: HCD_HandleTypeDef hhcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_HCD_Init() API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the HAL_HCD_MspInit() API:
   a. Enable the HCD/USB Low Level interface clock using the following macro
      – __HAL_RCC_USB_OTG_FS_CLK_ENABLE()
   b. Initialize the related GPIO clocks
   c. Configure HCD pin-out
   d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
   a. hhcd.pData = phost;
6. Enable HCD transmission and reception:
   a. HAL_HCD_Start();

32.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- *HAL_HCD_Init()*
- *HAL_HCD_HC_Init()*
- *HAL_HCD_HC_Halt()*
- *HAL_HCD_DeInit()*
- *HAL_HCD_MspInit()*
- *HAL_HCD_MspDeInit()*

32.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the USB Host Data Transfer

This section contains the following APIs:

- *HAL_HCD_HC_SubmitRequest()*
- *HAL_HCD_IRQHandler()*
- *HAL_HCD_SOF_Callback()*
- *HAL_HCD_Connect_Callback()*
- *HAL_HCD_Disconnect_Callback()*
- *HAL_HCD_HC_NotifyURBChange_Callback()*

32.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the HCD data transfers.

This section contains the following APIs:

- *HAL_HCD_Start()*
- *HAL_HCD_Stop()*
- *HAL_HCD_ResetPort()*

32.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_HCD_GetState()*
- *HAL_HCD_HC_GetURBState()*
- *HAL_HCD_HC_GetXferCount()*
- *HAL_HCD_HC_GetState()*
- *HAL_HCD_GetCurrentFrame()*
- *HAL_HCD_GetCurrentSpeed()*

32.2.6 Detailed description of functions

HAL_HCD_Init

Function name	**HAL_StatusTypeDef HAL_HCD_Init (HCD_HandleTypeDef * hhcd)**
Function description	Initialize the Host driver.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** status

HAL_HCD_DeInit

Function name	**HAL_StatusTypeDef HAL_HCD_DeInit (HCD_HandleTypeDef * hhcd)**
Function description	DeInitialize the Host driver.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** status

HAL_HCD_HC_Init

Function name	**HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)**
Function description	Initialize a Host channel.
Parameters	• **hhcd:** HCD handle • **ch_num:** Channel number. This parameter can be a value from 1 to 15 • **epnum:** Endpoint number. This parameter can be a value from 1 to 15 • **dev_address:** : Current device address This parameter can be a value from 0 to 255 • **speed:** Current device speed. This parameter can be one of these values: HCD_SPEED_HIGH: High speed mode, HCD_SPEED_FULL: Full speed mode, HCD_SPEED_LOW: Low speed mode • **ep_type:** Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type, EP_TYPE_ISOC: Isochronous type, EP_TYPE_BULK: Bulk type, EP_TYPE_INTR: Interrupt type • **mps:** Max Packet Size. This parameter can be a value from 0 to32K
Return values	• **HAL:** status

HAL_HCD_HC_Halt

Function name	**HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDef * hhcd, uint8_t ch_num)**
Function description	Halt a Host channel.
Parameters	• **hhcd:** HCD handle • **ch_num:** Channel number. This parameter can be a value from 1 to 15
Return values	• **HAL:** status

HAL_HCD_MspInit

Function name	**void HAL_HCD_MspInit (HCD_HandleTypeDef * hhcd)**
Function description	Initialize the HCD MSP.
Parameters	• **hhcd:** HCD handle

Return values	• **None:**

HAL_HCD_MspDeInit

Function name	**void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)**
Function description	DeInitialize the HCD MSP.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_HC_SubmitRequest

Function name	**HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)**
Function description	Submit a new URB for processing.
Parameters	• **hhcd:** HCD handle • **ch_num:** Channel number. This parameter can be a value from 1 to 15 • **direction:** Channel number. This parameter can be one of these values: 0: Output / 1: Input • **ep_type:** Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type/ EP_TYPE_ISOC: Isochronous type/ EP_TYPE_BULK: Bulk type/ EP_TYPE_INTR: Interrupt type/ • **token:** Endpoint Type. This parameter can be one of these values: 0: HC_PID_SETUP / 1: HC_PID_DATA1 • **pbuff:** pointer to URB data • **length:** Length of URB data • **do_ping:** activate do ping protocol (for high speed only). This parameter can be one of these values: 0: do ping inactive / 1: do ping active
Return values	• **HAL:** status

HAL_HCD_IRQHandler

Function name	**void HAL_HCD_IRQHandler (HCD_HandleTypeDef * hhcd)**
Function description	Handle HCD interrupt request.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_SOF_Callback

Function name	**void HAL_HCD_SOF_Callback (HCD_HandleTypeDef * hhcd)**
Function description	SOF callback.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_Connect_Callback

Function name	**void HAL_HCD_Connect_Callback (HCD_HandleTypeDef * hhcd)**
Function description	Connection Event callback.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_Disconnect_Callback

Function name	**void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)**
Function description	Disconnection Event callback.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_HC_NotifyURBChange_Callback

Function name	**void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDef * hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)**
Function description	Notify URB state change callback.
Parameters	• **hhcd:** HCD handle • **chnum:** Channel number. This parameter can be a value from 1 to 15 • **urb_state:** This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/
Return values	• **None:**

HAL_HCD_ResetPort

Function name	**HAL_StatusTypeDef HAL_HCD_ResetPort (HCD_HandleTypeDef * hhcd)**
Function description	Reset the Host port.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** status

HAL_HCD_Start

Function name	**HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDef * hhcd)**
Function description	Start the Host driver.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** status

HAL_HCD_Stop

Function name	**HAL_StatusTypeDef HAL_HCD_Stop (HCD_HandleTypeDef * hhcd)**
Function description	Stop the Host driver.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** status

HAL_HCD_GetState

Function name	**HCD_StateTypeDef HAL_HCD_GetState (HCD_HandleTypeDef * hhcd)**
Function description	Return the HCD handle state.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** state

HAL_HCD_HC_GetURBState

Function name	**HCD_URBStateTypeDef HAL_HCD_HC_GetURBState (HCD_HandleTypeDef * hhcd, uint8_t chnum)**
Function description	Return URB state for a channel.
Parameters	• **hhcd:** HCD handle • **chnum:** Channel number. This parameter can be a value from 1 to 15
Return values	• **URB:** state. This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL

HAL_HCD_HC_GetXferCount

Function name	**uint32_t HAL_HCD_HC_GetXferCount (HCD_HandleTypeDef * hhcd, uint8_t chnum)**
Function description	Return the last Host transfer size.
Parameters	• **hhcd:** HCD handle • **chnum:** Channel number. This parameter can be a value from 1 to 15
Return values	• **last:** transfer size in byte

HAL_HCD_HC_GetState

Function name	**HCD_HCStateTypeDef HAL_HCD_HC_GetState (HCD_HandleTypeDef * hhcd, uint8_t chnum)**
Function description	Return the Host Channel state.
Parameters	• **hhcd:** HCD handle • **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values	• **Host:** channel state This parameter can be one of these values: HC_IDLE/ HC_XFRC/ HC_HALTED/ HC_NYET/ HC_NAK/ HC_STALL/ HC_XACTERR/ HC_BBLERR/ HC_DATATGLERR

HAL_HCD_GetCurrentFrame

Function name	**uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)**
Function description	Return the current Host frame number.
Parameters	• **hhcd:** HCD handle
Return values	• **Current:** Host frame number

HAL_HCD_GetCurrentSpeed

Function name	**uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)**
Function description	Return the Host enumeration speed.
Parameters	• **hhcd:** HCD handle
Return values	• **Enumeration:** speed

32.3 HCD Firmware driver defines

32.3.1 HCD

HCD Exported Macros

__HAL_HCD_ENABLE

__HAL_HCD_DISABLE

__HAL_HCD_GET_FLAG

__HAL_HCD_CLEAR_FLAG

__HAL_HCD_IS_INVALID_INTERRUPT

__HAL_HCD_CLEAR_HC_INT

__HAL_HCD_MASK_HALT_HC_INT

__HAL_HCD_UNMASK_HALT_HC_INT

__HAL_HCD_MASK_ACK_HC_INT

__HAL_HCD_UNMASK_ACK_HC_INT

HCD PHY Module

HCD_PHY_EMBEDDED

HCD Speed

HCD_SPEED_HIGH

HCD_SPEED_LOW

HCD_SPEED_FULL

33 HAL I2C Generic Driver

33.1 I2C Firmware driver registers structures

33.1.1 I2C_InitTypeDef

Data Fields

- ***uint32_t Timing***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***

Field Documentation

- ***uint32_t I2C_InitTypeDef::Timing***
  Specifies the I2C_TIMINGR_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- ***uint32_t I2C_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t I2C_InitTypeDef::AddressingMode***
  Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of ***I2C_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***I2C_DUAL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t I2C_InitTypeDef::OwnAddress2Masks***
  Specifies the acknowledge mask address second device own address if dual addressing mode is selected This parameter can be a value of ***I2C_OWN_ADDRESS2_MASKS***
- ***uint32_t I2C_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***I2C_GENERAL_CALL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***I2C_NOSTRETCH_MODE***

33.1.2 __I2C_HandleTypeDef

Data Fields

- ***I2C_TypeDef * Instance***
- ***I2C_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***

- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_StatusTypeDef(* XferISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_I2C_StateTypeDef State***
- ***__IO HAL_I2C_ModeTypeDef Mode***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t AddrEventCount***

Field Documentation

- ***I2C_TypeDef* __I2C_HandleTypeDef::Instance***
  I2C registers base address
- ***I2C_InitTypeDef __I2C_HandleTypeDef::Init***
  I2C communication parameters
- ***uint8_t* __I2C_HandleTypeDef::pBuffPtr***
  Pointer to I2C transfer buffer
- ***uint16_t __I2C_HandleTypeDef::XferSize***
  I2C transfer size
- ***__IO uint16_t __I2C_HandleTypeDef::XferCount***
  I2C transfer counter
- ***__IO uint32_t __I2C_HandleTypeDef::XferOptions***
  I2C sequential transfer options, this parameter can be a value of ***I2C_XFEROPTIONS***
- ***__IO uint32_t __I2C_HandleTypeDef::PreviousState***
  I2C communication Previous state
- ***HAL_StatusTypeDef(* __I2C_HandleTypeDef::XferISR)(struct __I2C_HandleTypeDef *hi2c, uint32_t ITFlags, uint32_t ITSources)***
  I2C transfer IRQ handler function pointer
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmatx***
  I2C Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmarx***
  I2C Rx DMA handle parameters
- ***HAL_LockTypeDef __I2C_HandleTypeDef::Lock***
  I2C locking object
- ***__IO HAL_I2C_StateTypeDef __I2C_HandleTypeDef::State***
  I2C communication state
- ***__IO HAL_I2C_ModeTypeDef __I2C_HandleTypeDef::Mode***
  I2C communication mode
- ***__IO uint32_t __I2C_HandleTypeDef::ErrorCode***
  I2C Error code
- ***__IO uint32_t __I2C_HandleTypeDef::AddrEventCount***
  I2C Address Event counter

33.2 I2C Firmware driver API description

33.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C_HandleTypeDef handle structure, for example: I2C_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implementing the HAL_I2C_MspInit() API:
   a. Enable the I2Cx interface clock

   b. I2C pins configuration
      – Enable the clock for the I2C GPIOs
      – Configure I2C pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      – Configure the I2Cx interrupt priority
      – Enable the NVIC I2C IRQ Channel
   d. DMA Configuration if you need to use DMA process
      – Declare a DMA_HandleTypeDef handle structure for the transmit or receive channel
      – Enable the DMAx interface clock using
      – Configure the DMA handle parameters
      – Configure the DMA Tx or Rx channel
      – Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
      – Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel
3. Configure the Communication Clock Timing, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
4. Initialize the I2C registers by calling the HAL_I2C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I2C_MspInit(&hi2c) API.
5. To check if target device is ready for communication, use the function HAL_I2C_IsDeviceReady()
6. For I2C IO and IO MEM operations, three operation modes are available within this driver:

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Receive_IT()

- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Transmit_IT()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode IO sequential operation

These interfaces allow to manage a sequential transfer with a repeated start condition when a direction change during transfer

- A specific option field manage the different steps of a sequential transfer
- Option field values are defined through @ref I2C_XFEROPTIONS and are listed below:
  - I2C_FIRST_AND_LAST_FRAME: No sequential usage, functionnal is same as associated interfaces in no sequential mode
  - I2C_FIRST_FRAME: Sequential usage, this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition
  - I2C_FIRST_AND_NEXT_FRAME: Sequential usage (Master only), this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition, an then permit a call the same master sequential interface several times (like HAL_I2C_Master_Sequential_Transmit_IT() then HAL_I2C_Master_Sequential_Transmit_IT())
  - I2C_NEXT_FRAME: Sequential usage, this option allow to manage a sequence with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and without a final stop condition in both cases
  - I2C_LAST_FRAME: Sequential usage, this option allow to manage a sequance with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
- Differents sequential I2C interfaces are listed below:
  - Sequential transmit in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Sequential_Transmit_IT()

    - At transmission end of current frame transfer, HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
  - Sequential receive in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Sequential_Receive_IT()
    - At reception end of current frame transfer, HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
  - Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
    - End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()
  - Enable/disable the Address listen mode in slave I2C mode using HAL_I2C_EnableListen_IT() HAL_I2C_DisableListen_IT()
    - When address slave I2C match, HAL_I2C_AddrCallback() is executed and user can add his own code to check the Address Match Code and the transmission direction request by master (Write/Read).
    - At Listen mode end HAL_I2C_ListenCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_ListenCpltCallback()
  - Sequential transmit in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Sequential_Transmit_IT()
    - At transmission end of current frame transfer, HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
  - Sequential receive in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Sequential_Receive_IT()
    - At reception end of current frame transfer, HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
  - In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()
  - Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
  - End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()
  - Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode IO MEM operation

- Write an amount of data in non-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback()

- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()

DMA mode IO operation

- Transmit in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Receive_DMA()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Receive_DMA()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

DMA mode IO MEM operation

- Write an amount of data in non-blocking mode with DMA to a specific memory address using HAL_I2C_Mem_Write_DMA()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()

I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- __HAL_I2C_ENABLE: Enable the I2C peripheral
- __HAL_I2C_DISABLE: Disable the I2C peripheral

- __HAL_I2C_GENERATE_NACK: Generate a Non-Acknowledge I2C peripheral in Slave mode
- __HAL_I2C_GET_FLAG: Check whether the specified I2C flag is set or not
- __HAL_I2C_CLEAR_FLAG: Clear the specified I2C pending flag
- __HAL_I2C_ENABLE_IT: Enable the specified I2C interrupt
- __HAL_I2C_DISABLE_IT: Disable the specified I2C interrupt

You can refer to the I2C HAL driver header file for more useful macros

33.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the I2Cx peripheral:

- User must Implement HAL_I2C_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2C_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- ***HAL_I2C_Init()***
- ***HAL_I2C_DeInit()***
- ***HAL_I2C_MspInit()***
- ***HAL_I2C_MspDeInit()***

33.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

1. There are two modes of transfer:
   - Blocking mode: The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode: The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are:
   - HAL_I2C_Master_Transmit()
   - HAL_I2C_Master_Receive()
   - HAL_I2C_Slave_Transmit()
   - HAL_I2C_Slave_Receive()
   - HAL_I2C_Mem_Write()

  - HAL_I2C_Mem_Read()
  - HAL_I2C_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are:
   - HAL_I2C_Master_Transmit_IT()
   - HAL_I2C_Master_Receive_IT()
   - HAL_I2C_Slave_Transmit_IT()
   - HAL_I2C_Slave_Receive_IT()
   - HAL_I2C_Mem_Write_IT()
   - HAL_I2C_Mem_Read_IT()
4. No-Blocking mode functions with DMA are:
   - HAL_I2C_Master_Transmit_DMA()
   - HAL_I2C_Master_Receive_DMA()
   - HAL_I2C_Slave_Transmit_DMA()
   - HAL_I2C_Slave_Receive_DMA()
   - HAL_I2C_Mem_Write_DMA()
   - HAL_I2C_Mem_Read_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_I2C_MemTxCpltCallback()
   - HAL_I2C_MemRxCpltCallback()
   - HAL_I2C_MasterTxCpltCallback()
   - HAL_I2C_MasterRxCpltCallback()
   - HAL_I2C_SlaveTxCpltCallback()
   - HAL_I2C_SlaveRxCpltCallback()
   - HAL_I2C_ErrorCallback()

This section contains the following APIs:

- ***HAL_I2C_Master_Transmit()***
- ***HAL_I2C_Master_Receive()***
- ***HAL_I2C_Slave_Transmit()***
- ***HAL_I2C_Slave_Receive()***
- ***HAL_I2C_Master_Transmit_IT()***
- ***HAL_I2C_Master_Receive_IT()***
- ***HAL_I2C_Slave_Transmit_IT()***
- ***HAL_I2C_Slave_Receive_IT()***
- ***HAL_I2C_Master_Transmit_DMA()***
- ***HAL_I2C_Master_Receive_DMA()***
- ***HAL_I2C_Slave_Transmit_DMA()***
- ***HAL_I2C_Slave_Receive_DMA()***
- ***HAL_I2C_Mem_Write()***
- ***HAL_I2C_Mem_Read()***
- ***HAL_I2C_Mem_Write_IT()***
- ***HAL_I2C_Mem_Read_IT()***
- ***HAL_I2C_Mem_Write_DMA()***
- ***HAL_I2C_Mem_Read_DMA()***
- ***HAL_I2C_IsDeviceReady()***
- ***HAL_I2C_Master_Sequential_Transmit_IT()***
- ***HAL_I2C_Master_Sequential_Receive_IT()***
- ***HAL_I2C_Slave_Sequential_Transmit_IT()***
- ***HAL_I2C_Slave_Sequential_Receive_IT()***
- ***HAL_I2C_EnableListen_IT()***
- ***HAL_I2C_DisableListen_IT()***
- ***HAL_I2C_Master_Abort_IT()***

33.2.4 Peripheral State, Mode and Error functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_I2C_GetState()*
- *HAL_I2C_GetMode()*
- *HAL_I2C_GetError()*

33.2.5 Detailed description of functions

HAL_I2C_Init

Function name	**HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)**
Function description	Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and initialize the associated handle.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **HAL:** status

HAL_I2C_DeInit

Function name	**HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)**
Function description	DeInitialize the I2C peripheral.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **HAL:** status

HAL_I2C_MspInit

Function name	**void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)**
Function description	Initialize the I2C MSP.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_MspDeInit

Function name	**void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)**
Function description	DeInitialize the I2C MSP.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_Master_Transmit

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Transmit**

	(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function description	Transmits in master mode an amount of data in blocking mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Master_Receive

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receives in master mode an amount of data in blocking mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Slave_Transmit

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Transmits in slave mode an amount of data in blocking mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Slave_Receive

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive in slave mode an amount of data in blocking mode.

Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Mem_Write

Function name	**HAL_StatusTypeDef HAL_I2C_Mem_Write (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Write an amount of data in blocking mode to a specific memory address.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **MemAddress:** Internal memory address • **MemAddSize:** Size of internal memory address • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_Mem_Read

Function name	**HAL_StatusTypeDef HAL_I2C_Mem_Read (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Read an amount of data in blocking mode from a specific memory address.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **MemAddress:** Internal memory address • **MemAddSize:** Size of internal memory address • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_I2C_IsDeviceReady

Function name	**HAL_StatusTypeDef HAL_I2C_IsDeviceReady**

	(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)
Function description	Checks if target device is ready for communication.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **Trials:** Number of trials • **Timeout:** Timeout duration
Return values	• **HAL:** status
Notes	• This function is used with Memory devices

HAL_I2C_Master_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)**
Function description	Transmit in master mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Master_Receive_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)**
Function description	Receive in master mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Slave_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT**

	(I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)
Function description	Transmit in slave mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Slave_Receive_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)**
Function description	Receive in slave mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Mem_Write_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)**
Function description	Write an amount of data in non-blocking mode with Interrupt to a specific memory address.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **MemAddress:** Internal memory address • **MemAddSize:** Size of internal memory address • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Mem_Read_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)**
Function description	Read an amount of data in non-blocking mode with Interrupt from a specific memory address.

Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **MemAddress:** Internal memory address • **MemAddSize:** Size of internal memory address • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Master_Sequential_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Sequential_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**
Function description	Sequential transmit in master I2C mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options
Return values	• **HAL:** status
Notes	• This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Sequential_Receive_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Sequential_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**
Function description	Sequential receive in master I2C mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options
Return values	• **HAL:** status

Notes	• This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Sequential_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Sequential_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**
Function description	Sequential transmit in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options
Return values	• **HAL:** status
Notes	• This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Sequential_Receive_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Sequential_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**
Function description	Sequential receive in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options
Return values	• **HAL:** status
Notes	• This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EnableListen_IT

Function name	**HAL_StatusTypeDef HAL_I2C_EnableListen_IT (I2C_HandleTypeDef * hi2c)**
Function description	Enable the Address listen mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **HAL:** status

HAL_I2C_DisableListen_IT

Function name	**HAL_StatusTypeDef HAL_I2C_DisableListen_IT (I2C_HandleTypeDef * hi2c)**
Function description	Disable the Address listen mode with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C
Return values	• **HAL:** status

HAL_I2C_Master_Abort_IT

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Abort_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress)**
Function description	Abort a master I2C IT or DMA process communication with Interrupt.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface
Return values	• **HAL:** status

HAL_I2C_Master_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)**
Function description	Transmit in master mode an amount of data in non-blocking mode with DMA.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Master_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)**
Function description	Receive in master mode an amount of data in non-blocking mode with DMA.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call

	interface
	• **pData:** Pointer to data buffer
	• **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Slave_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)**
Function description	Transmit in slave mode an amount of data in non-blocking mode with DMA.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Slave_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)**
Function description	Receive in slave mode an amount of data in non-blocking mode with DMA.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Mem_Write_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)**
Function description	Write an amount of data in non-blocking mode with DMA to a specific memory address.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **MemAddress:** Internal memory address • **MemAddSize:** Size of internal memory address • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_I2C_Mem_Read_DMA

Function name	**HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)**
Function description	Reads an amount of data in non-blocking mode with DMA from a specific memory address.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **MemAddress:** Internal memory address • **MemAddSize:** Size of internal memory address • **pData:** Pointer to data buffer • **Size:** Amount of data to be read
Return values	• **HAL:** status

HAL_I2C_EV_IRQHandler

Function name	**void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef * hi2c)**
Function description	This function handles I2C event interrupt request.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_ER_IRQHandler

Function name	**void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)**
Function description	This function handles I2C error interrupt request.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_MasterTxCpltCallback

Function name	**void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function description	Master Tx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_MasterRxCpltCallback

Function name	**void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function description	Master Rx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_SlaveTxCpltCallback

Function name	**void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function description	Slave Tx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_SlaveRxCpltCallback

Function name	**void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function description	Slave Rx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_AddrCallback

Function name	**void HAL_I2C_AddrCallback (I2C_HandleTypeDef * hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)**
Function description	Slave Address Match callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • **TransferDirection:** Master request Transfer Direction (Write/Read), value of I2C Transfer Direction Master Point of View • **AddrMatchCode:** Address Match Code
Return values	• **None:**

HAL_I2C_ListenCpltCallback

Function name	**void HAL_I2C_ListenCpltCallback (I2C_HandleTypeDef * hi2c)**
Function description	Listen Complete callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_MemTxCpltCallback

Function name	**void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function description	Memory Tx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_MemRxCpltCallback

Function name	**void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function description	Memory Rx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_ErrorCallback

Function name	**void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)**
Function description	I2C error callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_AbortCpltCallback

Function name	**void HAL_I2C_AbortCpltCallback (I2C_HandleTypeDef * hi2c)**
Function description	I2C abort callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **None:**

HAL_I2C_GetState

Function name	**HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)**
Function description	Return the I2C handle state.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **HAL:** state

HAL_I2C_GetMode

Function name	**HAL_I2C_ModeTypeDef HAL_I2C_GetMode**

	(I2C_HandleTypeDef * hi2c)
Function description	Returns the I2C Master, Slave, Memory or no mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• **HAL:** mode

HAL_I2C_GetError

Function name	**uint32_t HAL_I2C_GetError (I2C_HandleTypeDef * hi2c)**
Function description	Return the I2C error code.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **I2C:** Error Code

33.3 I2C Firmware driver defines

33.3.1 I2C

I2C Addressing Mode

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

I2C Dual Addressing Mode

I2C_DUALADDRESS_DISABLE

I2C_DUALADDRESS_ENABLE

I2C Error Code definition

HAL_I2C_ERROR_NONE	No error
HAL_I2C_ERROR_BERR	BERR error
HAL_I2C_ERROR_ARLO	ARLO error
HAL_I2C_ERROR_AF	ACKF error
HAL_I2C_ERROR_OVR	OVR error
HAL_I2C_ERROR_DMA	DMA transfer error
HAL_I2C_ERROR_TIMEOUT	Timeout error
HAL_I2C_ERROR_SIZE	Size Management error

I2C Exported Macros

__HAL_I2C_RESET_HANDLE_STATE

Description:

- Reset I2C handle state.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_ENABLE_IT

Description:

- Enable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_DISABLE_IT

Description:

- Disable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_GET_IT_SOURCE

Description:

- Check whether the specified I2C interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the I2C

interrupt source to check. This parameter can be one of the following values:
- – I2C_IT_ERRI Errors interrupt enable
- – I2C_IT_TCI Transfer complete interrupt enable
- – I2C_IT_STOPI STOP detection interrupt enable
- – I2C_IT_NACKI NACK received interrupt enable
- – I2C_IT_ADDRI Address match interrupt enable
- – I2C_IT_RXI RX interrupt enable
- – I2C_IT_TXI TX interrupt enable

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_I2C_GET_FLAG

Description:

- Check whether the specified I2C flag is set or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - – I2C_FLAG_TXE Transmit data register empty
  - – I2C_FLAG_TXIS Transmit interrupt status
  - – I2C_FLAG_RXNE Receive data register not empty
  - – I2C_FLAG_ADDR Address matched (slave mode)
  - – I2C_FLAG_AF Acknowledge failure received flag
  - – I2C_FLAG_STOPF STOP detection flag
  - – I2C_FLAG_TC Transfer complete (master mode)
  - – I2C_FLAG_TCR Transfer complete reload
  - – I2C_FLAG_BERR Bus error
  - – I2C_FLAG_ARLO Arbitration lost
  - – I2C_FLAG_OVR Overrun/Underrun
  - – I2C_FLAG_PECERR PEC error in reception
  - – I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - – I2C_FLAG_ALERT SMBus alert
  - – I2C_FLAG_BUSY Bus busy
  - – I2C_FLAG_DIR Transfer direction

(slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_I2C_CLEAR_FLAG

Description:

- Clear the I2C pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_I2C_ENABLE

Description:

- Enable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_DISABLE

Description:

- Disable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_GENERATE_NACK

Description:

- Generate a Non-Acknowledge I2C peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

I2C Flag definition

I2C_FLAG_TXE

I2C_FLAG_TXIS

I2C_FLAG_RXNE

I2C_FLAG_ADDR

I2C_FLAG_AF

I2C_FLAG_STOPF

I2C_FLAG_TC

I2C_FLAG_TCR

I2C_FLAG_BERR

I2C_FLAG_ARLO

I2C_FLAG_OVR

I2C_FLAG_PECERR

I2C_FLAG_TIMEOUT

I2C_FLAG_ALERT

I2C_FLAG_BUSY

I2C_FLAG_DIR

I2C General Call Addressing Mode

I2C_GENERALCALL_DISABLE

I2C_GENERALCALL_ENABLE

I2C Interrupt configuration definition

I2C_IT_ERRI

I2C_IT_TCI

I2C_IT_STOPI

I2C_IT_NACKI

I2C_IT_ADDRI

I2C_IT_RXI

I2C_IT_TXI

I2C Memory Address Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

I2C No-Stretch Mode

I2C_NOSTRETCH_DISABLE

I2C_NOSTRETCH_ENABLE

I2C Own Address2 Masks

I2C_OA2_NOMASK

I2C_OA2_MASK01

I2C_OA2_MASK02

I2C_OA2_MASK03

I2C_OA2_MASK04

I2C_OA2_MASK05

I2C_OA2_MASK06

I2C_OA2_MASK07

I2C Reload End Mode

I2C_RELOAD_MODE

I2C_AUTOEND_MODE

I2C_SOFTEND_MODE

I2C Start or Stop Mode

I2C_NO_STARTSTOP

I2C_GENERATE_STOP

I2C_GENERATE_START_READ

I2C_GENERATE_START_WRITE

I2C Transfer Direction Master Point of View

I2C_DIRECTION_TRANSMIT

I2C_DIRECTION_RECEIVE

I2C Sequential Transfer Options

I2C_FIRST_FRAME

I2C_FIRST_AND_NEXT_FRAME

I2C_NEXT_FRAME

I2C_FIRST_AND_LAST_FRAME

I2C_LAST_FRAME

34 HAL I2C Extension Driver

34.1 I2CEx Firmware driver API description

34.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32L4xx devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop modes

34.1.2 How to use this driver

This driver provides functions to configure Noise Filter and Wake Up Feature

1. Configure I2C Analog noise filter using the function HAL_I2CEx_ConfigAnalogFilter()
2. Configure I2C Digital noise filter using the function HAL_I2CEx_ConfigDigitalFilter()
3. Configure the enable or disable of I2C Wake Up Mode using the functions:
   - HAL_I2CEx_EnableWakeUp()
   - HAL_I2CEx_DisableWakeUp()
4. Configure the enable or disable of fast mode plus driving capability using the functions:
   - HAL_I2CEx_EnableFastModePlus()
   - HAL_I2CEx_DisableFastModePlus()

34.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters
- Configure Wake Up Feature

This section contains the following APIs:

- ***HAL_I2CEx_ConfigAnalogFilter()***
- ***HAL_I2CEx_ConfigDigitalFilter()***
- ***HAL_I2CEx_EnableWakeUp()***
- ***HAL_I2CEx_DisableWakeUp()***
- ***HAL_I2CEx_EnableFastModePlus()***
- ***HAL_I2CEx_DisableFastModePlus()***

34.1.4 Detailed description of functions

HAL_I2CEx_ConfigAnalogFilter

Function name	**HAL_StatusTypeDef HAL_I2CEx_ConfigAnalogFilter (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)**
Function description	Configure I2C Analog noise filter.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral. • **AnalogFilter:** New state of the Analog filter.

Return values	• **HAL:** status

HAL_I2CEx_ConfigDigitalFilter

Function name	**HAL_StatusTypeDef HAL_I2CEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)**
Function description	Configure I2C Digital noise filter.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral. • **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.
Return values	• **HAL:** status

HAL_I2CEx_EnableWakeUp

Function name	**HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)**
Function description	Enable I2C wakeup from stop mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	• **HAL:** status

HAL_I2CEx_DisableWakeUp

Function name	**HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)**
Function description	Disable I2C wakeup from stop mode.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	• **HAL:** status

HAL_I2CEx_EnableFastModePlus

Function name	**void HAL_I2CEx_EnableFastModePlus (uint32_t ConfigFastModePlus)**
Function description	Enable the I2C fast mode plus driving capability.
Parameters	• **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values
Return values	• **None:**
Notes	• For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9. • For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C1 parameter. • For all I2C2 pins fast mode plus driving capability can be

enabled only by using I2C_FASTMODEPLUS_I2C2 parameter.
- For all I2C3 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C3 parameter.
- For all I2C4 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C4 parameter.

HAL_I2CEx_DisableFastModePlus

Function name	**void HAL_I2CEx_DisableFastModePlus (uint32_t ConfigFastModePlus)**
Function description	Disable the I2C fast mode plus driving capability.
Parameters	• **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values
Return values	• **None:**
Notes	• For I2C1, fast mode plus driving capability can be disabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9. • For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C1 parameter. • For all I2C2 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C2 parameter. • For all I2C3 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C3 parameter. • For all I2C4 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C4 parameter.

34.2 I2CEx Firmware driver defines

34.2.1 I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2C Extended Fast Mode Plus

I2C_FMP_NOT_SUPPORTED	Fast Mode Plus not supported
I2C_FASTMODEPLUS_PB6	Enable Fast Mode Plus on PB6
I2C_FASTMODEPLUS_PB7	Enable Fast Mode Plus on PB7
I2C_FASTMODEPLUS_PB8	Enable Fast Mode Plus on PB8
I2C_FASTMODEPLUS_PB9	Enable Fast Mode Plus on PB9

I2C_FASTMODEPLUS_I2C1	Enable Fast Mode Plus on I2C1 pins
I2C_FASTMODEPLUS_I2C2	Enable Fast Mode Plus on I2C2 pins
I2C_FASTMODEPLUS_I2C3	Enable Fast Mode Plus on I2C3 pins
I2C_FASTMODEPLUS_I2C4	Enable Fast Mode Plus on I2C4 pins

35 HAL IRDA Generic Driver

35.1 IRDA Firmware driver registers structures

35.1.1 IRDA_InitTypeDef

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint8_t Prescaler***
- ***uint16_t PowerMode***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate***
  This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hirda->Init.BaudRate))) where usart_ker_ckpres is the IRDA input clock divided by a prescaler
- ***uint32_t IRDA_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***IRDA_Word_Length***
- ***uint32_t IRDA_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***IRDA_Parity***
  Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t IRDA_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***IRDA_Transfer_Mode***
- ***uint8_t IRDA_InitTypeDef::Prescaler***
  Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.
  Note: Prescaler value 0 is forbidden
- ***uint16_t IRDA_InitTypeDef::PowerMode***
  Specifies the IRDA power mode. This parameter can be a value of ***IRDA_Low_Power***
- ***uint32_t IRDA_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the IRDA clock source. This parameter can be a value of ***IRDA_ClockPrescaler***.

35.1.2 IRDA_HandleTypeDef

Data Fields

- ***USART_TypeDef * Instance***
- ***IRDA_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***

- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IRDA_StateTypeDef gState***
- ***__IO HAL_IRDA_StateTypeDef RxState***
- ***uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* IRDA_HandleTypeDef::Instance***
  USART registers base address
- ***IRDA_InitTypeDef IRDA_HandleTypeDef::Init***
  IRDA communication parameters
- ***uint8_t* IRDA_HandleTypeDef::pTxBuffPtr***
  Pointer to IRDA Tx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::TxXferSize***
  IRDA Tx Transfer size
- ***__IO uint16_t IRDA_HandleTypeDef::TxXferCount***
  IRDA Tx Transfer Counter
- ***uint8_t* IRDA_HandleTypeDef::pRxBuffPtr***
  Pointer to IRDA Rx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::RxXferSize***
  IRDA Rx Transfer size
- ***__IO uint16_t IRDA_HandleTypeDef::RxXferCount***
  IRDA Rx Transfer Counter
- ***uint16_t IRDA_HandleTypeDef::Mask***
  USART RX RDR register mask
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmatx***
  IRDA Tx DMA Handle parameters
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmarx***
  IRDA Rx DMA Handle parameters
- ***HAL_LockTypeDef IRDA_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::gState***
  IRDA state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::RxState***
  IRDA state information related to Rx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***uint32_t IRDA_HandleTypeDef::ErrorCode***
  IRDA Error code

35.2 IRDA Firmware driver API description

35.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA_HandleTypeDef handle structure (eg. IRDA_HandleTypeDef hirda).
2. Initialize the IRDA low level resources by implementing the HAL_IRDA_MspInit() API in setting the associated USART or UART in IRDA mode:
   – Enable the USARTx/UARTx interface clock.

- USARTx/UARTx pins configuration:
  - Enable the clock for the USARTx/UARTx GPIOs.
  - Configure these USARTx/UARTx pins (TX as alternate function pull-up, RX as alternate function Input).
- NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
  - Configure the USARTx/UARTx interrupt priority.
  - Enable the NVIC USARTx/UARTx IRQ handle.
  - The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.
- DMA Configuration if you need to use DMA process (HAL_IRDA_Transmit_DMA() and HAL_IRDA_Receive_DMA() APIs):
  - Declare a DMA handle structure for the Tx/Rx channel.
  - Enable the DMAx interface clock.
  - Configure the declared DMA handle structure with the required Tx/Rx parameters.
  - Configure the DMA Tx/Rx channel.
  - Associate the initialized DMA handle to the IRDA DMA Tx/Rx handle.
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.

3. Program the Baud Rate, Word Length and Parity and Mode(Receiver/Transmitter), the normal or low power mode and the clock prescaler in the hirda handle Init structure.
4. Initialize the IRDA registers by calling the HAL_IRDA_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_IRDA_MspInit() API. The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.
5. Three operation modes are available within this driver:

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_IRDA_Transmit()
- Receive an amount of data in blocking mode using HAL_IRDA_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_IRDA_Transmit_IT()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_IRDA_Receive_IT()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_IRDA_Transmit_DMA()
- At transmission half of transfer HAL_IRDA_TxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxHalfCpltCallback()

- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_IRDA_Receive_DMA()
- At reception half of transfer HAL_IRDA_RxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxHalfCpltCallback()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

IRDA HAL driver macros list

Below the list of most used macros in IRDA HAL driver.

- __HAL_IRDA_ENABLE: Enable the IRDA peripheral
- __HAL_IRDA_DISABLE: Disable the IRDA peripheral
- __HAL_IRDA_GET_FLAG: Check whether the specified IRDA flag is set or not
- __HAL_IRDA_CLEAR_FLAG: Clear the specified IRDA pending flag
- __HAL_IRDA_ENABLE_IT: Enable the specified IRDA interrupt
- __HAL_IRDA_DISABLE_IT: Disable the specified IRDA interrupt
- __HAL_IRDA_GET_IT_SOURCE: Check whether or not the specified IRDA interrupt is enabled

You can refer to the IRDA HAL driver header file for more useful macros

35.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx in asynchronous IRDA mode.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Power mode
  - Prescaler setting
  - Receiver/transmitter modes

The HAL_IRDA_Init() API follows the USART asynchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_IRDA_Init()***
- ***HAL_IRDA_DeInit()***
- ***HAL_IRDA_MspInit()***
- ***HAL_IRDA_MspDeInit()***

35.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the IRDA data transfers.

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
   - Blocking mode: the communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - Non-Blocking mode: the communication is performed using Interrupts or DMA, these APIs return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are:
   - HAL_IRDA_Transmit()
   - HAL_IRDA_Receive()
3. Non Blocking mode APIs with Interrupt are:
   - HAL_IRDA_Transmit_IT()
   - HAL_IRDA_Receive_IT()
   - HAL_IRDA_IRQHandler()
4. Non Blocking mode functions with DMA are:
   - HAL_IRDA_Transmit_DMA()
   - HAL_IRDA_Receive_DMA()
   - HAL_IRDA_DMAPause()
   - HAL_IRDA_DMAResume()
   - HAL_IRDA_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non Blocking mode:
   - HAL_IRDA_TxHalfCpltCallback()
   - HAL_IRDA_TxCpltCallback()
   - HAL_IRDA_RxHalfCpltCallback()
   - HAL_IRDA_RxCpltCallback()
   - HAL_IRDA_ErrorCallback()
6. Non-Blocking mode transfers could be aborted using Abort APIs:
   - HAL_IRDA_Abort()
   - HAL_IRDA_AbortTransmit()
   - HAL_IRDA_AbortReceive()
   - HAL_IRDA_Abort_IT()
   - HAL_IRDA_AbortTransmit_IT()
   - AL_IRDA_AbortReceive_IT()
7. For Abort services based on interrupts (HAL_IRDA_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
   - HAL_IRDA_AbortCpltCallback()
   - HAL_IRDA_AbortTransmitCpltCallback()
   - HAL_IRDA_AbortReceiveCpltCallback()
8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows:
   - Error is considered as Recoverable and non-blocking: Transfer could go till end, but error severity is to be evaluated by user: this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception. Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed. Transfer is kept

ongoing on IRDA side. If user wants to abort it, Abort services should be called by user.

– Error is considered as Blocking: Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_IRDA_Transmit()***
- ***HAL_IRDA_Receive()***
- ***HAL_IRDA_Transmit_IT()***
- ***HAL_IRDA_Receive_IT()***
- ***HAL_IRDA_Transmit_DMA()***
- ***HAL_IRDA_Receive_DMA()***
- ***HAL_IRDA_DMAPause()***
- ***HAL_IRDA_DMAResume()***
- ***HAL_IRDA_DMAStop()***
- ***HAL_IRDA_Abort()***
- ***HAL_IRDA_AbortTransmit()***
- ***HAL_IRDA_AbortReceive()***
- ***HAL_IRDA_Abort_IT()***
- ***HAL_IRDA_AbortTransmit_IT()***
- ***HAL_IRDA_AbortReceive_IT()***
- ***HAL_IRDA_IRQHandler()***
- ***HAL_IRDA_TxCpltCallback()***
- ***HAL_IRDA_TxHalfCpltCallback()***
- ***HAL_IRDA_RxCpltCallback()***
- ***HAL_IRDA_RxHalfCpltCallback()***
- ***HAL_IRDA_ErrorCallback()***
- ***HAL_IRDA_AbortCpltCallback()***
- ***HAL_IRDA_AbortTransmitCpltCallback()***
- ***HAL_IRDA_AbortReceiveCpltCallback()***

35.2.4 Peripheral State and Error functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process

- HAL_IRDA_GetState() API can be helpful to check in run-time the state of the IRDA peripheral handle.
- HAL_IRDA_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- ***HAL_IRDA_GetState()***
- ***HAL_IRDA_GetError()***

35.2.5 Detailed description of functions

HAL_IRDA_Init

Function name	**HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)**

Function description	Initialize the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and initialize the associated handle.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **HAL:** status

HAL_IRDA_DeInit

Function name	**HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)**
Function description	DeInitialize the IRDA peripheral.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **HAL:** status

HAL_IRDA_MspInit

Function name	**void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)**
Function description	Initialize the IRDA MSP.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None:**

HAL_IRDA_MspDeInit

Function name	**void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)**
Function description	DeInitialize the IRDA MSP.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None:**

HAL_IRDA_Transmit

Function name	**HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Send an amount of data in blocking mode.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be sent. • **Timeout:** Specify timeout value.

Return values	• **HAL:** status

HAL_IRDA_Receive

Function name	**HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be received. • **Timeout:** Specify timeout value.
Return values	• **HAL:** status

HAL_IRDA_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**
Function description	Send an amount of data in interrupt mode.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be sent.
Return values	• **HAL:** status

HAL_IRDA_Receive_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in interrupt mode.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be received.
Return values	• **HAL:** status

HAL_IRDA_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**
Function description	Send an amount of data in DMA mode.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values
- **HAL:** status

HAL_IRDA_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in DMA mode.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be received.
Return values	• **HAL:** status
Notes	• When the IRDA parity is enabled (PCE = 1), the received data contains the parity bit (MSB position).

HAL_IRDA_DMAPause

Function name	**HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)**
Function description	Pause the DMA Transfer.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **HAL:** status

HAL_IRDA_DMAResume

Function name	**HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)**
Function description	Resume the DMA Transfer.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status

HAL_IRDA_DMAStop

Function name	**HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)**
Function description	Stop the DMA Transfer.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values	• **HAL:** status

HAL_IRDA_Abort

Function name	**HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing transfers (blocking mode).
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode: when exiting function, Abort is considered as completed.

HAL_IRDA_AbortTransmit

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortTransmit (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Transmit transfer (blocking mode).
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode: when exiting function, Abort is considered as completed.

HAL_IRDA_AbortReceive

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortReceive (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Receive transfer (blocking mode).
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx

- transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode: when exiting function, Abort is considered as completed.

HAL_IRDA_Abort_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_Abort_IT (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing transfers (Interrupt mode).
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortTransmit_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortTransmit_IT (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Transmit transfer (Interrupt mode).
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortReceive_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortReceive_IT (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Receive transfer (Interrupt mode).
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_IRQHandler

Function name	**void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)**
Function description	Handle IRDA interrupt request.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None:**

HAL_IRDA_TxCpltCallback

Function name	**void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	Tx Transfer completed callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None:**

HAL_IRDA_RxCpltCallback

Function name	**void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	Rx Transfer completed callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values	• **None:**

HAL_IRDA_TxHalfCpltCallback

Function name	**void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	Tx Half Transfer completed callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• **None:**

HAL_IRDA_RxHalfCpltCallback

Function name	**void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	Rx Half Transfer complete callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None:**

HAL_IRDA_ErrorCallback

Function name	**void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA error callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None:**

HAL_IRDA_AbortCpltCallback

Function name	**void HAL_IRDA_AbortCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA Abort Complete callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None:**

HAL_IRDA_AbortTransmitCpltCallback

Function name	**void HAL_IRDA_AbortTransmitCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA Abort Complete callback.

Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None:**

HAL_IRDA_AbortReceiveCpltCallback

Function name	**void HAL_IRDA_AbortReceiveCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA Abort Receive Complete callback.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None:**

HAL_IRDA_GetState

Function name	**HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)**
Function description	Return the IRDA handle state.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **HAL:** state

HAL_IRDA_GetError

Function name	**uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)**
Function description	Return the IRDA handle error code.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **IRDA:** Error Code

35.3 IRDA Firmware driver defines

35.3.1 IRDA

Clock Prescaler

IRDA_PRESCALER_DIV1	fclk_pres = fclk
IRDA_PRESCALER_DIV2	fclk_pres = fclk/2
IRDA_PRESCALER_DIV4	fclk_pres = fclk/4
IRDA_PRESCALER_DIV6	fclk_pres = fclk/6
IRDA_PRESCALER_DIV8	fclk_pres = fclk/8
IRDA_PRESCALER_DIV10	fclk_pres = fclk/10

IRDA_PRESCALER_DIV12	fclk_pres = fclk/12
IRDA_PRESCALER_DIV16	fclk_pres = fclk/16
IRDA_PRESCALER_DIV32	fclk_pres = fclk/32
IRDA_PRESCALER_DIV64	fclk_pres = fclk/64
IRDA_PRESCALER_DIV128	fclk_pres = fclk/128
IRDA_PRESCALER_DIV256	fclk_pres = fclk/256

IRDA DMA Rx

IRDA_DMA_RX_DISABLE	IRDA DMA RX disabled
IRDA_DMA_RX_ENABLE	IRDA DMA RX enabled

IRDA DMA Tx

IRDA_DMA_TX_DISABLE	IRDA DMA TX disabled
IRDA_DMA_TX_ENABLE	IRDA DMA TX enabled

IRDA Exported Macros

__HAL_IRDA_RESET_HANDLE_STATE

Description:

- Reset IRDA handle state.

Parameters:

- __HANDLE__: IRDA handle.

Return value:

- None

__HAL_IRDA_FLUSH_DRREGISTER

Description:

- Flush the IRDA DR register.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FLAG

Description:

- Clear the specified IRDA pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - IRDA_CLEAR_PEF
  - IRDA_CLEAR_FEF
  - IRDA_CLEAR_NEF
  - IRDA_CLEAR_OREF
  - IRDA_CLEAR_TCF

– IRDA_CLEAR_IDLEF

Return value:

- None

__HAL_IRDA_CLEAR_PEFLAG

Description:

- Clear the IRDA PE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FEFLAG

Description:

- Clear the IRDA FE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_NEFLAG

Description:

- Clear the IRDA NE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_OREFLAG

Description:

- Clear the IRDA ORE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_IDLEFLAG

Description:

- Clear the IRDA IDLE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_GET_FLAG

Description:

- Check whether the specified IRDA flag is set or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - IRDA_FLAG_REACK Receive enable acknowledge flag
  - IRDA_FLAG_TEACK Transmit enable acknowledge flag
  - IRDA_FLAG_BUSY Busy flag
  - IRDA_FLAG_ABRF Auto Baud rate detection flag
  - IRDA_FLAG_ABRE Auto Baud rate detection error flag
  - IRDA_FLAG_TXE Transmit data register empty flag
  - IRDA_FLAG_TC Transmission Complete flag
  - IRDA_FLAG_RXNE Receive data register not empty flag
  - IRDA_FLAG_ORE OverRun Error flag
  - IRDA_FLAG_NE Noise Error flag
  - IRDA_FLAG_FE Framing Error flag
  - IRDA_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_IRDA_ENABLE_IT

Description:

- Enable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt

- IRDA_IT_PE Parity Error interrupt
- IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_DISABLE_IT

Description:

- Disable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_GET_IT

Description:

- Check whether the specified IRDA interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ORE OverRun Error interrupt
  - IRDA_IT_NE Noise Error interrupt
  - IRDA_IT_FE Framing Error interrupt

- IRDA_IT_PE Parity Error interrupt

Return value:

- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_GET_IT_SOURCE

Description:

- Check whether the specified IRDA interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ERR Framing, overrun or noise error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:

- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_CLEAR_IT

Description:

- Clear the specified IRDA ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - IRDA_CLEAR_PEF Parity Error Clear Flag
  - IRDA_CLEAR_FEF Framing Error Clear Flag
  - IRDA_CLEAR_NEF Noise detected Clear Flag
  - IRDA_CLEAR_OREF OverRun Error Clear Flag
  - IRDA_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_IRDA_SEND_REQ

Description:

- Set a specific IRDA request flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - IRDA_AUTOBAUD_REQUEST Auto-Baud Rate Request
  - IRDA_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - IRDA_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ENABLE

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_DISABLE	**Description:** • Disable UART/USART associated to IRDA Handle. **Parameters:** • __HANDLE__: specifies the IRDA Handle. **Return value:** • None

IRDA Flags

IRDA_FLAG_REACK	IRDA receive enable acknowledge flag
IRDA_FLAG_TEACK	IRDA transmit enable acknowledge flag
IRDA_FLAG_BUSY	IRDA busy flag
IRDA_FLAG_ABRF	IRDA auto Baud rate flag
IRDA_FLAG_ABRE	IRDA auto Baud rate error
IRDA_FLAG_TXE	IRDA transmit data register empty
IRDA_FLAG_TC	IRDA transmission complete
IRDA_FLAG_RXNE	IRDA read data register not empty
IRDA_FLAG_ORE	IRDA overrun error
IRDA_FLAG_NE	IRDA noise error
IRDA_FLAG_FE	IRDA frame error
IRDA_FLAG_PE	IRDA parity error

IRDA interruptions flags mask

IRDA_IT_MASK	IRDA Interruptions flags mask

IRDA Interrupts Definition

IRDA_IT_PE	IRDA Parity error interruption
IRDA_IT_TXE	IRDA Transmit data register empty interruption
IRDA_IT_TC	IRDA Transmission complete interruption
IRDA_IT_RXNE	IRDA Read data register not empty interruption
IRDA_IT_IDLE	IRDA Idle interruption
IRDA_IT_ERR	IRDA Error interruption
IRDA_IT_ORE	IRDA Overrun error interruption
IRDA_IT_NE	IRDA Noise error interruption
IRDA_IT_FE	IRDA Frame error interruption

IRDA Interruption Clear Flags

IRDA_CLEAR_PEF	Parity Error Clear Flag
IRDA_CLEAR_FEF	Framing Error Clear Flag
IRDA_CLEAR_NEF	Noise detected Clear Flag

IRDA_CLEAR_OREF	OverRun Error Clear Flag
IRDA_CLEAR_IDLEF	IDLE line detected Clear Flag
IRDA_CLEAR_TCF	Transmission Complete Clear Flag

IRDA Low Power

IRDA_POWERMODE_NORMAL	IRDA normal power mode
IRDA_POWERMODE_LOWPOWER	IRDA low power mode

IRDA Mode

IRDA_MODE_DISABLE	Associated UART disabled in IRDA mode
IRDA_MODE_ENABLE	Associated UART enabled in IRDA mode

IRDA One Bit Sampling

IRDA_ONE_BIT_SAMPLE_DISABLE	One-bit sampling disabled
IRDA_ONE_BIT_SAMPLE_ENABLE	One-bit sampling enabled

IRDA Parity

IRDA_PARITY_NONE	No parity
IRDA_PARITY_EVEN	Even parity
IRDA_PARITY_ODD	Odd parity

IRDA Request Parameters

IRDA_AUTOBAUD_REQUEST	Auto-Baud Rate Request
IRDA_RXDATA_FLUSH_REQUEST	Receive Data flush Request
IRDA_TXDATA_FLUSH_REQUEST	Transmit data flush Request

IRDA State

IRDA_STATE_DISABLE	IRDA disabled
IRDA_STATE_ENABLE	IRDA enabled

IRDA Transfer Mode

IRDA_MODE_RX	RX mode
IRDA_MODE_TX	TX mode
IRDA_MODE_TX_RX	RX and TX mode

IRDA Word Length

IRDA_WORDLENGTH_7B	7-bit long frame
IRDA_WORDLENGTH_8B	8-bit long frame
IRDA_WORDLENGTH_9B	9-bit long frame

36 HAL IWDG Generic Driver

36.1 IWDG Firmware driver registers structures

36.1.1 IWDG_InitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Reload***
- ***uint32_t Window***

Field Documentation

- ***uint32_t IWDG_InitTypeDef::Prescaler***
  Select the prescaler of the IWDG. This parameter can be a value of ***IWDG_Prescaler***
- ***uint32_t IWDG_InitTypeDef::Reload***
  Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::Window***
  Specifies the window value to be compared to the down-counter. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF

36.1.2 IWDG_HandleTypeDef

Data Fields

- ***IWDG_TypeDef * Instance***
- ***IWDG_InitTypeDef Init***

Field Documentation

- ***IWDG_TypeDef* IWDG_HandleTypeDef::Instance***
  Register base address
- ***IWDG_InitTypeDef IWDG_HandleTypeDef::Init***
  IWDG required parameters

36.2 IWDG Firmware driver API description

36.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by Low-Speed clock (LSI) and thus stays active even if the main clock fails.
- Once the IWDG is started, the LSI is forced ON and both can not be disabled. The counter starts counting down from the reset value (0xFFF). When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0x0000 AAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded in the counter and the watchdog reset is prevented.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake-up from STANDBY). IWDGRST flag in RCC_CSR register can be used to inform when an IWDG reset occurs.
- Debug mode: When the microcontroller enters debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP

configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_IWDG() and __HAL_DBGMCU_UNFREEZE_IWDG() macros

Min-max timeout value @32KHz (LSI): ~125us / ~32.7s The IWDG timeout may vary due to LSI frequency dispersion. STM32L4xx devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

36.2.2 How to use this driver

1. Use IWDG using HAL_IWDG_Init() function to:
   - Enable instance by writing Start keyword in IWDG_KEY register. LSI clock is forced ON and IWDG counter starts downcounting.
   - Enable write access to configuration register: IWDG_PR, IWDG_RLR & IWDG_WINR.
   - Configure the IWDG prescaler and counter reload value. This reload value will be loaded in the IWDG counter each time the watchdog is reloaded, then the IWDG will start counting down from this value.
   - Wait for status flags to be reset
   - Depending on window parameter:
     - If Window Init parameter is same as Window register value, nothing more is done but reload counter value in order to exit function withy exact time base.
     - Else modify Window register. This will automatically reload watchdog counter.
2. Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_IWDG_Refresh() function.

IWDG HAL driver macros list

Below the list of most used macros in IWDG HAL driver:

- __HAL_IWDG_START: Enable the IWDG peripheral
- __HAL_IWDG_RELOAD_COUNTER: Reloads IWDG counter with value defined in the reload register

36.2.3 Initialization and Start functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef of associated handle.
- Manage Window option.
- Once initialization is performed in HAL_IWDG_Init function, Watchdog is reloaded in order to exit function with correct time base.

This section contains the following APIs:

- ***HAL_IWDG_Init()***

36.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the IWDG.

This section contains the following APIs:

- ***HAL_IWDG_Refresh()***

36.2.5 Detailed description of functions

HAL_IWDG_Init

Function name	**HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)**
Function description	Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and start watchdog.
Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• **HAL:** status

HAL_IWDG_Refresh

Function name	**HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)**
Function description	Refresh the IWDG.
Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• **HAL:** status

36.3 IWDG Firmware driver defines

36.3.1 IWDG

IWDG Exported Macros

__HAL_IWDG_START	**Description:** • Enable the IWDG peripheral. **Parameters:** • __HANDLE__: IWDG handle **Return value:** • None
__HAL_IWDG_RELOAD_COUNTER	**Description:** • Reload IWDG counter with value defined in the reload register (write access to IWDG_PR, IWDG_RLR & IWDG_WINR registers disabled). **Parameters:** • __HANDLE__: IWDG handle **Return value:** • None

IWDG Prescaler

IWDG_PRESCALER_4	IWDG prescaler set to 4
IWDG_PRESCALER_8	IWDG prescaler set to 8
IWDG_PRESCALER_16	IWDG prescaler set to 16
IWDG_PRESCALER_32	IWDG prescaler set to 32
IWDG_PRESCALER_64	IWDG prescaler set to 64
IWDG_PRESCALER_128	IWDG prescaler set to 128
IWDG_PRESCALER_256	IWDG prescaler set to 256

IWDG Window option

IWDG_WINDOW_DISABLE

37 HAL LCD Generic Driver

37.1 LCD Firmware driver registers structures

37.1.1 LCD_InitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Divider***
- ***uint32_t Duty***
- ***uint32_t Bias***
- ***uint32_t VoltageSource***
- ***uint32_t Contrast***
- ***uint32_t DeadTime***
- ***uint32_t PulseOnDuration***
- ***uint32_t HighDrive***
- ***uint32_t BlinkMode***
- ***uint32_t BlinkFrequency***
- ***uint32_t MuxSegment***

Field Documentation

- ***uint32_t LCD_InitTypeDef::Prescaler***
  Configures the LCD Prescaler. This parameter can be one value of ***LCD_Prescaler***
- ***uint32_t LCD_InitTypeDef::Divider***
  Configures the LCD Divider. This parameter can be one value of ***LCD_Divider***
- ***uint32_t LCD_InitTypeDef::Duty***
  Configures the LCD Duty. This parameter can be one value of ***LCD_Duty***
- ***uint32_t LCD_InitTypeDef::Bias***
  Configures the LCD Bias. This parameter can be one value of ***LCD_Bias***
- ***uint32_t LCD_InitTypeDef::VoltageSource***
  Selects the LCD Voltage source. This parameter can be one value of ***LCD_Voltage_Source***
- ***uint32_t LCD_InitTypeDef::Contrast***
  Configures the LCD Contrast. This parameter can be one value of ***LCD_Contrast***
- ***uint32_t LCD_InitTypeDef::DeadTime***
  Configures the LCD Dead Time. This parameter can be one value of ***LCD_DeadTime***
- ***uint32_t LCD_InitTypeDef::PulseOnDuration***
  Configures the LCD Pulse On Duration. This parameter can be one value of ***LCD_PulseOnDuration***
- ***uint32_t LCD_InitTypeDef::HighDrive***
  Enable or disable the low resistance divider. This parameter can be one value of ***LCD_HighDrive***
- ***uint32_t LCD_InitTypeDef::BlinkMode***
  Configures the LCD Blink Mode. This parameter can be one value of ***LCD_BlinkMode***
- ***uint32_t LCD_InitTypeDef::BlinkFrequency***
  Configures the LCD Blink frequency. This parameter can be one value of ***LCD_BlinkFrequency***
- ***uint32_t LCD_InitTypeDef::MuxSegment***
  Enable or disable mux segment. This parameter can be one value of ***LCD_MuxSegment***

37.1.2 LCD_HandleTypeDef

Data Fields

- ***LCD_TypeDef * Instance***
- ***LCD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***LCD_TypeDef* LCD_HandleTypeDef::Instance***
- ***LCD_InitTypeDef LCD_HandleTypeDef::Init***
- ***HAL_LockTypeDef LCD_HandleTypeDef::Lock***
- ***__IO HAL_LCD_StateTypeDef LCD_HandleTypeDef::State***
- ***__IO uint32_t LCD_HandleTypeDef::ErrorCode***

37.2 LCD Firmware driver API description

37.2.1 How to use this driver

The LCD HAL driver can be used as follows:

1. Declare a LCD_HandleTypeDef handle structure. The frequency generator allows you to achieve various LCD frame rates starting from an LCD input clock frequency (LCDCLK) which can vary from 32 kHz up to 1 MHz.
2. Initialize the LCD low level resources by implementing the HAL_LCD_MspInit() API:
   - Enable the LCDCLK (same as RTCCLK): to configure the RTCCLK/LCDCLK, proceed as follows:
     - Use RCC function HAL_RCCEx_PeriphCLKConfig in indicating RCC_PERIPHCLK_LCD and selected clock source (HSE, LSI or LSE)
   - LCD pins configuration:
     - Enable the clock for the LCD GPIOs.
     - Configure these LCD pins as alternate function no-pull.
   - Enable the LCD interface clock.
3. Program the Prescaler, Divider, Blink mode, Blink Frequency Duty, Bias, Voltage Source, Dead Time, Pulse On Duration, Contrast, High drive and Multiplexer Segment in the Init structure of the LCD handle.
4. Initialize the LCD registers by calling the HAL_LCD_Init() API. The HAL_LCD_Init() API configures also the low level Hardware GPIO, CLOCK, ...etc) by calling the customized HAL_LCD_MspInit() API. After calling the HAL_LCD_Init() the LCD RAM memory is cleared
5. Optionally you can update the LCD configuration using these macros:
   - LCD High Drive using the __HAL_LCD_HIGHDRIVER_ENABLE() and __HAL_LCD_HIGHDRIVER_DISABLE() macros
   - Voltage output buffer using __HAL_LCD_VOLTAGE_BUFFER_ENABLE() and __HAL_LCD_VOLTAGE_BUFFER_DISABLE() macros
   - LCD Pulse ON Duration using the __HAL_LCD_PULSEONDURATION_CONFIG() macro
   - LCD Dead Time using the __HAL_LCD_DEADTIME_CONFIG() macro
   - The LCD Blink mode and frequency using the __HAL_LCD_BLINK_CONFIG() macro
   - The LCD Contrast using the __HAL_LCD_CONTRAST_CONFIG() macro

6. Write to the LCD RAM memory using the HAL_LCD_Write() API, this API can be called more time to update the different LCD RAM registers before calling HAL_LCD_UpdateDisplayRequest() API.
7. The HAL_LCD_Clear() API can be used to clear the LCD RAM memory.
8. When LCD RAM memory is updated enable the update display request using the HAL_LCD_UpdateDisplayRequest() API.

LCD and low power modes:

1. The LCD remain active during Sleep, Low Power run, Low Power Sleep and STOP modes.

37.2.2 Initialization and Configuration functions

This section contains the following APIs:

- ***HAL_LCD_Init()***
- ***HAL_LCD_DeInit()***
- ***HAL_LCD_MspDeInit()***
- ***HAL_LCD_MspInit()***

37.2.3 IO operation functions

Using its double buffer memory the LCD controller ensures the coherency of the displayed information without having to use interrupts to control LCD_RAM modification. The application software can access the first buffer level (LCD_RAM) through the APB interface. Once it has modified the LCD_RAM using the HAL_LCD_Write() API, it sets the UDR flag in the LCD_SR register using the HAL_LCD_UpdateDisplayRequest() API. This UDR flag (update display request) requests the updated information to be moved into the second buffer level (LCD_DISPLAY). This operation is done synchronously with the frame (at the beginning of the next frame), until the update is completed, the LCD_RAM is write protected and the UDR flag stays high. Once the update is completed another flag (UDD - Update Display Done) is set and generates an interrupt if the UDDIE bit in the LCD_FCR register is set. The time it takes to update LCD_DISPLAY is, in the worst case, one odd and one even frame. The update will not occur (UDR = 1 and UDD = 0) until the display is enabled (LCDEN = 1).

This section contains the following APIs:

- ***HAL_LCD_Write()***
- ***HAL_LCD_Clear()***
- ***HAL_LCD_UpdateDisplayRequest()***

37.2.4 Peripheral State functions

This subsection provides a set of functions allowing to control the LCD:

- HAL_LCD_GetState() API can be helpful to check in run-time the state of the LCD peripheral State.
- HAL_LCD_GetError() API to return the LCD error code.

This section contains the following APIs:

- ***HAL_LCD_GetState()***
- ***HAL_LCD_GetError()***

37.2.5 Detailed description of functions

HAL_LCD_DeInit

Function name	**HAL_StatusTypeDef HAL_LCD_DeInit (LCD_HandleTypeDef * hlcd)**
Function description	DeInitialize the LCD peripheral.
Parameters	• **hlcd:** LCD handle
Return values	• **HAL:** status

HAL_LCD_Init

Function name	**HAL_StatusTypeDef HAL_LCD_Init (LCD_HandleTypeDef * hlcd)**
Function description	Initialize the LCD peripheral according to the specified parameters in the LCD_InitStruct and initialize the associated handle.
Parameters	• **hlcd:** LCD handle
Return values	• **None:**
Notes	• This function can be used only when the LCD is disabled.

HAL_LCD_MspInit

Function name	**void HAL_LCD_MspInit (LCD_HandleTypeDef * hlcd)**
Function description	Initialize the LCD MSP.
Parameters	• **hlcd:** LCD handle
Return values	• **None:**

HAL_LCD_MspDeInit

Function name	**void HAL_LCD_MspDeInit (LCD_HandleTypeDef * hlcd)**
Function description	DeInitialize the LCD MSP.
Parameters	• **hlcd:** LCD handle
Return values	• **None:**

HAL_LCD_Write

Function name	**HAL_StatusTypeDef HAL_LCD_Write (LCD_HandleTypeDef * hlcd, uint32_t RAMRegisterIndex, uint32_t RAMRegisterMask, uint32_t Data)**
Function description	Write a word in the specific LCD RAM.
Parameters	• **hlcd:** LCD handle • **RAMRegisterIndex:** specifies the LCD RAM Register. This parameter can be one of the following values: – LCD_RAM_REGISTER0: LCD RAM Register 0 – LCD_RAM_REGISTER1: LCD RAM Register 1 – LCD_RAM_REGISTER2: LCD RAM Register 2

- LCD_RAM_REGISTER3: LCD RAM Register 3
- LCD_RAM_REGISTER4: LCD RAM Register 4
- LCD_RAM_REGISTER5: LCD RAM Register 5
- LCD_RAM_REGISTER6: LCD RAM Register 6
- LCD_RAM_REGISTER7: LCD RAM Register 7
- LCD_RAM_REGISTER8: LCD RAM Register 8
- LCD_RAM_REGISTER9: LCD RAM Register 9
- LCD_RAM_REGISTER10: LCD RAM Register 10
- LCD_RAM_REGISTER11: LCD RAM Register 11
- LCD_RAM_REGISTER12: LCD RAM Register 12
- LCD_RAM_REGISTER13: LCD RAM Register 13
- LCD_RAM_REGISTER14: LCD RAM Register 14
- LCD_RAM_REGISTER15: LCD RAM Register 15

- **RAMRegisterMask:** specifies the LCD RAM Register Data Mask.
- **Data:** specifies LCD Data Value to be written.

Return values
- **None:**

HAL_LCD_Clear

Function name	**HAL_StatusTypeDef HAL_LCD_Clear (LCD_HandleTypeDef * hlcd)**
Function description	Clear the LCD RAM registers.
Parameters	• **hlcd:** LCD handle
Return values	• **None:**

HAL_LCD_UpdateDisplayRequest

Function name	**HAL_StatusTypeDef HAL_LCD_UpdateDisplayRequest (LCD_HandleTypeDef * hlcd)**
Function description	Enable the Update Display Request.
Parameters	• **hlcd:** LCD handle
Return values	• **None:**
Notes	• Each time software modifies the LCD_RAM it must set the UDR bit to transfer the updated data to the second level buffer. The UDR bit stays set until the end of the update and during this time the LCD_RAM is write protected. • When the display is disabled, the update is performed for all LCD_DISPLAY locations. When the display is enabled, the update is performed only for locations for which commons are active (depending on DUTY). For example if DUTY = 1/2, only the LCD_DISPLAY of COM0 and COM1 will be updated.

HAL_LCD_GetState

Function name	**HAL_LCD_StateTypeDef HAL_LCD_GetState (LCD_HandleTypeDef * hlcd)**
Function description	Return the LCD handle state.

Parameters	• **hlcd:** LCD handle
Return values	• **HAL:** state

HAL_LCD_GetError

Function name	**uint32_t HAL_LCD_GetError (LCD_HandleTypeDef * hlcd)**
Function description	Return the LCD error code.
Parameters	• **hlcd:** LCD handle
Return values	• **LCD:** Error Code

LCD_WaitForSynchro

Function name	**HAL_StatusTypeDef LCD_WaitForSynchro (LCD_HandleTypeDef * hlcd)**
Function description	Wait until the LCD FCR register is synchronized in the LCDCLK domain.
Return values	• **None:**

37.3 LCD Firmware driver defines

37.3.1 LCD

LCD Bias

LCD_BIAS_1_4	1/4 Bias
LCD_BIAS_1_2	1/2 Bias
LCD_BIAS_1_3	1/3 Bias

LCD Blink Frequency

LCD_BLINKFREQUENCY_DIV8	The Blink frequency = fLCD/8
LCD_BLINKFREQUENCY_DIV16	The Blink frequency = fLCD/16
LCD_BLINKFREQUENCY_DIV32	The Blink frequency = fLCD/32
LCD_BLINKFREQUENCY_DIV64	The Blink frequency = fLCD/64
LCD_BLINKFREQUENCY_DIV128	The Blink frequency = fLCD/128
LCD_BLINKFREQUENCY_DIV256	The Blink frequency = fLCD/256
LCD_BLINKFREQUENCY_DIV512	The Blink frequency = fLCD/512
LCD_BLINKFREQUENCY_DIV1024	The Blink frequency = fLCD/1024

LCD Blink Mode

LCD_BLINKMODE_OFF	Blink disabled
LCD_BLINKMODE_SEG0_COM0	Blink enabled on SEG[0], COM[0] (1 pixel)
LCD_BLINKMODE_SEG0_ALLCOM	Blink enabled on SEG[0], all COM (up to 8 pixels according to the programmed duty)
LCD_BLINKMODE_ALLSEG_ALLCOM	Blink enabled on all SEG and all COM (all pixels)

LCD Contrast

LCD_CONTRASTLEVEL_0	Maximum Voltage = 2.60V
LCD_CONTRASTLEVEL_1	Maximum Voltage = 2.73V
LCD_CONTRASTLEVEL_2	Maximum Voltage = 2.86V
LCD_CONTRASTLEVEL_3	Maximum Voltage = 2.99V
LCD_CONTRASTLEVEL_4	Maximum Voltage = 3.12V
LCD_CONTRASTLEVEL_5	Maximum Voltage = 3.26V
LCD_CONTRASTLEVEL_6	Maximum Voltage = 3.40V
LCD_CONTRASTLEVEL_7	Maximum Voltage = 3.55V

LCD Dead Time

LCD_DEADTIME_0	No dead Time
LCD_DEADTIME_1	One Phase between different couple of Frame
LCD_DEADTIME_2	Two Phase between different couple of Frame
LCD_DEADTIME_3	Three Phase between different couple of Frame
LCD_DEADTIME_4	Four Phase between different couple of Frame
LCD_DEADTIME_5	Five Phase between different couple of Frame
LCD_DEADTIME_6	Six Phase between different couple of Frame
LCD_DEADTIME_7	Seven Phase between different couple of Frame

LCD Divider

LCD_DIVIDER_16	LCD frequency = CLKPS/16
LCD_DIVIDER_17	LCD frequency = CLKPS/17
LCD_DIVIDER_18	LCD frequency = CLKPS/18
LCD_DIVIDER_19	LCD frequency = CLKPS/19
LCD_DIVIDER_20	LCD frequency = CLKPS/20
LCD_DIVIDER_21	LCD frequency = CLKPS/21
LCD_DIVIDER_22	LCD frequency = CLKPS/22
LCD_DIVIDER_23	LCD frequency = CLKPS/23
LCD_DIVIDER_24	LCD frequency = CLKPS/24
LCD_DIVIDER_25	LCD frequency = CLKPS/25
LCD_DIVIDER_26	LCD frequency = CLKPS/26
LCD_DIVIDER_27	LCD frequency = CLKPS/27
LCD_DIVIDER_28	LCD frequency = CLKPS/28
LCD_DIVIDER_29	LCD frequency = CLKPS/29
LCD_DIVIDER_30	LCD frequency = CLKPS/30
LCD_DIVIDER_31	LCD frequency = CLKPS/31

LCD Duty

LCD_DUTY_STATIC	Static duty
LCD_DUTY_1_2	1/2 duty
LCD_DUTY_1_3	1/3 duty
LCD_DUTY_1_4	1/4 duty
LCD_DUTY_1_8	1/8 duty

LCD Error Code

HAL_LCD_ERROR_NONE	No error
HAL_LCD_ERROR_FCRSF	Synchro flag timeout error
HAL_LCD_ERROR_UDR	Update display request flag timeout error
HAL_LCD_ERROR_UDD	Update display done flag timeout error
HAL_LCD_ERROR_ENS	LCD enabled status flag timeout error
HAL_LCD_ERROR_RDY	LCD Booster ready timeout error

LCD Exported Macros

__HAL_LCD_RESET_HANDLE_STATE

Description:

- Reset LCD handle state.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_ENABLE

Description:

- Enable the LCD peripheral.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_DISABLE

Description:

- Disable the LCD peripheral.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_HIGHDRIVER_ENABLE

Description:

- Enable the low resistance divider.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

Notes:

- Displays with high internal resistance may need a longer drive time to achieve satisfactory contrast. This function is useful in this case if some additional power consumption can be tolerated. When this mode is enabled, the PulseOn Duration (PON) have to be programmed to 1/CK_PS (LCD_PULSEONDURATION_1).

__HAL_LCD_HIGHDRIVER_DISABLE

Description:

- Disable the low resistance divider.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_VOLTAGE_BUFFER_ENABLE

Description:

- Enable the voltage output buffer for higher driving capability.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_VOLTAGE_BUFFER_DISABLE

Description:

- Disable the voltage output buffer for higher driving capability.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_PULSEONDURATION_CONFIG

Description:

- Configure the LCD pulse on duration.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __DURATION__: specifies the LCD pulse on duration in terms of CK_PS (prescaled LCD clock period) pulses. This parameter can be one of the following values:
  - LCD_PULSEONDURATION_0: 0 pulse
  - LCD_PULSEONDURATION_1: Pulse ON duration = 1/CK_PS

  - LCD_PULSEONDURATION_2: Pulse ON duration = 2/CK_PS
  - LCD_PULSEONDURATION_3: Pulse ON duration = 3/CK_PS
  - LCD_PULSEONDURATION_4: Pulse ON duration = 4/CK_PS
  - LCD_PULSEONDURATION_5: Pulse ON duration = 5/CK_PS
  - LCD_PULSEONDURATION_6: Pulse ON duration = 6/CK_PS
  - LCD_PULSEONDURATION_7: Pulse ON duration = 7/CK_PS

Return value:

- None

__HAL_LCD_DEADTIME_CONFIG

Description:

- Configure the LCD dead time.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __DEADTIME__: specifies the LCD dead time. This parameter can be one of the following values:
  - LCD_DEADTIME_0: No dead Time
  - LCD_DEADTIME_1: One Phase between different couple of Frame
  - LCD_DEADTIME_2: Two Phase between different couple of Frame
  - LCD_DEADTIME_3: Three Phase between different couple of Frame
  - LCD_DEADTIME_4: Four Phase between different couple of Frame
  - LCD_DEADTIME_5: Five Phase between different couple of Frame
  - LCD_DEADTIME_6: Six Phase between different couple of Frame
  - LCD_DEADTIME_7: Seven Phase between different couple of Frame

Return value:

- None

__HAL_LCD_CONTRAST_CONFIG

Description:

- Configure the LCD contrast.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __CONTRAST__: specifies the LCD Contrast. This parameter can be one of the following values:
  - LCD_CONTRASTLEVEL_0: Maximum Voltage = 2.60V
  - LCD_CONTRASTLEVEL_1: Maximum

Voltage = 2.73V
- LCD_CONTRASTLEVEL_2: Maximum Voltage = 2.86V
- LCD_CONTRASTLEVEL_3: Maximum Voltage = 2.99V
- LCD_CONTRASTLEVEL_4: Maximum Voltage = 3.12V
- LCD_CONTRASTLEVEL_5: Maximum Voltage = 3.25V
- LCD_CONTRASTLEVEL_6: Maximum Voltage = 3.38V
- LCD_CONTRASTLEVEL_7: Maximum Voltage = 3.51V

Return value:

- None

__HAL_LCD_BLINK_CONFIG

Description:

- Configure the LCD Blink mode and Blink frequency.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __BLINKMODE__: specifies the LCD blink mode. This parameter can be one of the following values:
  - LCD_BLINKMODE_OFF: Blink disabled
  - LCD_BLINKMODE_SEG0_COM0: Blink enabled on SEG[0], COM[0] (1 pixel)
  - LCD_BLINKMODE_SEG0_ALLCOM: Blink enabled on SEG[0], all COM (up to 8 pixels according to the programmed duty)
  - LCD_BLINKMODE_ALLSEG_ALLCOM: Blink enabled on all SEG and all COM (all pixels)
- __BLINKFREQUENCY__: specifies the LCD blink frequency.
  - LCD_BLINKFREQUENCY_DIV8: The Blink frequency = fLcd/8
  - LCD_BLINKFREQUENCY_DIV16: The Blink frequency = fLcd/16
  - LCD_BLINKFREQUENCY_DIV32: The Blink frequency = fLcd/32
  - LCD_BLINKFREQUENCY_DIV64: The Blink frequency = fLcd/64
  - LCD_BLINKFREQUENCY_DIV128: The Blink frequency = fLcd/128
  - LCD_BLINKFREQUENCY_DIV256: The Blink frequency = fLcd/256
  - LCD_BLINKFREQUENCY_DIV512: The Blink frequency = fLcd/512
  - LCD_BLINKFREQUENCY_DIV1024: The Blink frequency = fLcd/1024

Return value:

- None

__HAL_LCD_ENABLE_IT

Description:

- Enable the specified LCD interrupt.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __INTERRUPT__: specifies the LCD interrupt source to be enabled. This parameter can be one of the following values:
  - LCD_IT_SOF: Start of Frame Interrupt
  - LCD_IT_UDD: Update Display Done Interrupt

Return value:

- None

__HAL_LCD_DISABLE_IT

Description:

- Disable the specified LCD interrupt.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __INTERRUPT__: specifies the LCD interrupt source to be disabled. This parameter can be one of the following values:
  - LCD_IT_SOF: Start of Frame Interrupt
  - LCD_IT_UDD: Update Display Done Interrupt

Return value:

- None

__HAL_LCD_GET_IT_SOURCE

Description:

- Check whether the specified LCD interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __IT__: specifies the LCD interrupt source to check. This parameter can be one of the following values:
  - LCD_IT_SOF: Start of Frame Interrupt
  - LCD_IT_UDD: Update Display Done Interrupt.

Return value:

- The: state of __IT__ (TRUE or FALSE).

Notes:

- If the device is in STOP mode (PCLK not provided) UDD will not generate an interrupt even if UDDIE = 1. If the display is not enabled the UDD interrupt will never occur.

__HAL_LCD_GET_FLAG

Description:

- Check whether the specified LCD flag is set or not.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - LCD_FLAG_ENS: LCD Enabled flag. It indicates the LCD controller status.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

Notes:

- The ENS bit is set immediately when the LCDEN bit in the LCD_CR goes from 0 to 1. On deactivation it reflects the real status of LCD so it becomes 0 at the end of the last displayed frame. LCD_FLAG_SOF: Start of Frame flag. This flag is set by hardware at the beginning of a new frame, at the same time as the display data is updated. LCD_FLAG_UDR: Update Display Request flag. LCD_FLAG_UDD: Update Display Done flag. LCD_FLAG_RDY: Step_up converter Ready flag. It indicates the status of the step-up converter. LCD_FLAG_FCRSF: LCD Frame Control Register Synchronization Flag. This flag is set by hardware each time the LCD_FCR register is updated in the LCDCLK domain.

__HAL_LCD_CLEAR_FLAG

Description:

- Clear the specified LCD pending flag.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - LCD_FLAG_SOF: Start of Frame Interrupt
  - LCD_FLAG_UDD: Update Display Done Interrupt

Return value:

- None

LCD Flags Definition

LCD_FLAG_ENS	LCD enabled status
LCD_FLAG_SOF	Start of frame flag
LCD_FLAG_UDR	Update display request

LCD_FLAG_UDD	Update display done
LCD_FLAG_RDY	Ready flag
LCD_FLAG_FCRSF	LCD Frame Control Register Synchronization flag

LCD High Drive

LCD_HIGHDRIVE_DISABLE	High drive disabled
LCD_HIGHDRIVE_ENABLE	High drive enabled

LCD Interrupts

LCD_IT_SOF

LCD_IT_UDD

LCD Mux Segment

LCD_MUXSEGMENT_DISABLE	SEG pin multiplexing disabled
LCD_MUXSEGMENT_ENABLE	SEG[31:28] are multiplexed with SEG[43:40]

LCD Prescaler

LCD_PRESCALER_1	CLKPS = LCDCLK
LCD_PRESCALER_2	CLKPS = LCDCLK/2
LCD_PRESCALER_4	CLKPS = LCDCLK/4
LCD_PRESCALER_8	CLKPS = LCDCLK/8
LCD_PRESCALER_16	CLKPS = LCDCLK/16
LCD_PRESCALER_32	CLKPS = LCDCLK/32
LCD_PRESCALER_64	CLKPS = LCDCLK/64
LCD_PRESCALER_128	CLKPS = LCDCLK/128
LCD_PRESCALER_256	CLKPS = LCDCLK/256
LCD_PRESCALER_512	CLKPS = LCDCLK/512
LCD_PRESCALER_1024	CLKPS = LCDCLK/1024
LCD_PRESCALER_2048	CLKPS = LCDCLK/2048
LCD_PRESCALER_4096	CLKPS = LCDCLK/4096
LCD_PRESCALER_8192	CLKPS = LCDCLK/8192
LCD_PRESCALER_16384	CLKPS = LCDCLK/16384
LCD_PRESCALER_32768	CLKPS = LCDCLK/32768

LCD Pulse On Duration

LCD_PULSEONDURATION_0	Pulse ON duration = 0 pulse
LCD_PULSEONDURATION_1	Pulse ON duration = 1/CK_PS
LCD_PULSEONDURATION_2	Pulse ON duration = 2/CK_PS
LCD_PULSEONDURATION_3	Pulse ON duration = 3/CK_PS
LCD_PULSEONDURATION_4	Pulse ON duration = 4/CK_PS
LCD_PULSEONDURATION_5	Pulse ON duration = 5/CK_PS

LCD_PULSEONDURATION_6	Pulse ON duration = 6/CK_PS
LCD_PULSEONDURATION_7	Pulse ON duration = 7/CK_PS

LCD RAMRegister

LCD_RAM_REGISTER0	LCD RAM Register 0
LCD_RAM_REGISTER1	LCD RAM Register 1
LCD_RAM_REGISTER2	LCD RAM Register 2
LCD_RAM_REGISTER3	LCD RAM Register 3
LCD_RAM_REGISTER4	LCD RAM Register 4
LCD_RAM_REGISTER5	LCD RAM Register 5
LCD_RAM_REGISTER6	LCD RAM Register 6
LCD_RAM_REGISTER7	LCD RAM Register 7
LCD_RAM_REGISTER8	LCD RAM Register 8
LCD_RAM_REGISTER9	LCD RAM Register 9
LCD_RAM_REGISTER10	LCD RAM Register 10
LCD_RAM_REGISTER11	LCD RAM Register 11
LCD_RAM_REGISTER12	LCD RAM Register 12
LCD_RAM_REGISTER13	LCD RAM Register 13
LCD_RAM_REGISTER14	LCD RAM Register 14
LCD_RAM_REGISTER15	LCD RAM Register 15

LCD Voltage Source

LCD_VOLTAGESOURCE_INTERNAL	Internal voltage source for the LCD
LCD_VOLTAGESOURCE_EXTERNAL	External voltage source for the LCD

38 HAL LPTIM Generic Driver

38.1 LPTIM Firmware driver registers structures

38.1.1 LPTIM_ClockConfigTypeDef

Data Fields

- ***uint32_t Source***
- ***uint32_t Prescaler***

Field Documentation

- ***uint32_t LPTIM_ClockConfigTypeDef::Source***
  Selects the clock source. This parameter can be a value of ***LPTIM_Clock_Source***
- ***uint32_t LPTIM_ClockConfigTypeDef::Prescaler***
  Specifies the counter clock Prescaler. This parameter can be a value of ***LPTIM_Clock_Prescaler***

38.1.2 LPTIM_ULPClockConfigTypeDef

Data Fields

- ***uint32_t Polarity***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_ULPClockConfigTypeDef::Polarity***
  Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of ***LPTIM_Clock_Polarity***
- ***uint32_t LPTIM_ULPClockConfigTypeDef::SampleTime***
  Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of ***LPTIM_Clock_Sample_Time***

38.1.3 LPTIM_TriggerConfigTypeDef

Data Fields

- ***uint32_t Source***
- ***uint32_t ActiveEdge***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_TriggerConfigTypeDef::Source***
  Selects the Trigger source. This parameter can be a value of ***LPTIM_Trigger_Source***
- ***uint32_t LPTIM_TriggerConfigTypeDef::ActiveEdge***
  Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_External_Trigger_Polarity***
- ***uint32_t LPTIM_TriggerConfigTypeDef::SampleTime***
  Selects the trigger sampling time to configure the clock glitch filter. Note: This

parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_Trigger_Sample_Time***

38.1.4 LPTIM_InitTypeDef

Data Fields

- ***LPTIM_ClockConfigTypeDef Clock***
- ***LPTIM_ULPClockConfigTypeDef UltraLowPowerClock***
- ***LPTIM_TriggerConfigTypeDef Trigger***
- ***uint32_t OutputPolarity***
- ***uint32_t UpdateMode***
- ***uint32_t CounterSource***
- ***uint32_t Input1Source***
- ***uint32_t Input2Source***

Field Documentation

- ***LPTIM_ClockConfigTypeDef LPTIM_InitTypeDef::Clock***
  Specifies the clock parameters
- ***LPTIM_ULPClockConfigTypeDef LPTIM_InitTypeDef::UltraLowPowerClock***
  Specifies the Ultra Low Power clock parameters
- ***LPTIM_TriggerConfigTypeDef LPTIM_InitTypeDef::Trigger***
  Specifies the Trigger parameters
- ***uint32_t LPTIM_InitTypeDef::OutputPolarity***
  Specifies the Output polarity. This parameter can be a value of ***LPTIM_Output_Polarity***
- ***uint32_t LPTIM_InitTypeDef::UpdateMode***
  Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of ***LPTIM_Updating_Mode***
- ***uint32_t LPTIM_InitTypeDef::CounterSource***
  Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of ***LPTIM_Counter_Source***
- ***uint32_t LPTIM_InitTypeDef::Input1Source***
  Specifies source selected for input1 (GPIO or comparator output). This parameter can be a value of ***LPTIM_Input1_Source***
- ***uint32_t LPTIM_InitTypeDef::Input2Source***
  Specifies source selected for input2 (GPIO or comparator output). Note: This parameter is used only for encoder feature so is used only for LPTIM1 instance. This parameter can be a value of ***LPTIM_Input2_Source***

38.1.5 LPTIM_HandleTypeDef

Data Fields

- ***LPTIM_TypeDef * Instance***
- ***LPTIM_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LPTIM_StateTypeDef State***

Field Documentation

- ***LPTIM_TypeDef* LPTIM_HandleTypeDef::Instance***
  Register base address
- ***LPTIM_InitTypeDef LPTIM_HandleTypeDef::Init***
  LPTIM required parameters

- ***HAL_StatusTypeDef LPTIM_HandleTypeDef::Status***
  LPTIM peripheral status
- ***HAL_LockTypeDef LPTIM_HandleTypeDef::Lock***
  LPTIM locking object
- ***__IO HAL_LPTIM_StateTypeDef LPTIM_HandleTypeDef::State***
  LPTIM peripheral state

38.2 LPTIM Firmware driver API description

38.2.1 How to use this driver

The LPTIM HAL driver can be used as follows:

1. Initialize the LPTIM low level resources by implementing the HAL_LPTIM_MspInit():
   - Enable the LPTIM interface clock using __HAL_RCC_LPTIMx_CLK_ENABLE().
   - In case of using interrupts (e.g. HAL_LPTIM_PWM_Start_IT()):
     - Configure the LPTIM interrupt priority using HAL_NVIC_SetPriority().
     - Enable the LPTIM IRQ handler using HAL_NVIC_EnableIRQ().
     - In LPTIM IRQ handler, call HAL_LPTIM_IRQHandler().
2. Initialize the LPTIM HAL using HAL_LPTIM_Init(). This function configures mainly:
   - The instance: LPTIM1 or LPTIM2.
   - Clock: the counter clock.
     - Source: it can be either the ULPTIM input (IN1) or one of the internal clock; (APB, LSE, LSI or MSI).
     - Prescaler: select the clock divider.
   - UltraLowPowerClock: To be used only if the ULPTIM is selected as counter clock source.
     - Polarity: polarity of the active edge for the counter unit if the ULPTIM input is selected.
     - SampleTime: clock sampling time to configure the clock glitch filter.
   - Trigger: How the counter start.
     - Source: trigger can be software or one of the hardware triggers.
     - ActiveEdge: only for hardware trigger.
     - SampleTime: trigger sampling time to configure the trigger glitch filter.
   - OutputPolarity: 2 opposite polarities are possible.
   - UpdateMode: specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period.
   - Input1Source: Source selected for input1 (GPIO or comparator output).
   - Input2Source: Source selected for input2 (GPIO or comparator output). Input2 is used only for encoder feature so is used only for LPTIM1 instance.
3. Six modes are available:
   - PWM Mode: To generate a PWM signal with specified period and pulse, call HAL_LPTIM_PWM_Start() or HAL_LPTIM_PWM_Start_IT() for interruption mode.
   - One Pulse Mode: To generate pulse with specified width in response to a stimulus, call HAL_LPTIM_OnePulse_Start() or HAL_LPTIM_OnePulse_Start_IT() for interruption mode.
   - Set once Mode: In this mode, the output changes the level (from low level to high level if the output polarity is configured high, else the opposite) when a compare match occurs. To start this mode, call HAL_LPTIM_SetOnce_Start() or HAL_LPTIM_SetOnce_Start_IT() for interruption mode.
   - Encoder Mode: To use the encoder interface call HAL_LPTIM_Encoder_Start() or HAL_LPTIM_Encoder_Start_IT() for interruption mode. Only available for LPTIM1 instance.

  - Time out Mode: an active edge on one selected trigger input rests the counter. The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. To start this mode call HAL_LPTIM_TimeOut_Start_IT() or HAL_LPTIM_TimeOut_Start_IT() for interruption mode.
  - Counter Mode: counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. To start this mode, call HAL_LPTIM_Counter_Start() or HAL_LPTIM_Counter_Start_IT() for interruption mode.
4. User can stop any process by calling the corresponding API: HAL_LPTIM_Xxx_Stop() or HAL_LPTIM_Xxx_Stop_IT() if the process is already started in interruption mode.
5. De-initialize the LPTIM peripheral using HAL_LPTIM_DeInit().

38.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.
- DeInitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- DeInitialize the LPTIM MSP.

This section contains the following APIs:

- ***HAL_LPTIM_Init()***
- ***HAL_LPTIM_DeInit()***
- ***HAL_LPTIM_MspInit()***
- ***HAL_LPTIM_MspDeInit()***

38.2.3 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.
- Stop the Set once mode.
- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.

This section contains the following APIs:

- ***HAL_LPTIM_PWM_Start()***
- ***HAL_LPTIM_PWM_Stop()***
- ***HAL_LPTIM_PWM_Start_IT()***
- ***HAL_LPTIM_PWM_Stop_IT()***
- ***HAL_LPTIM_OnePulse_Start()***
- ***HAL_LPTIM_OnePulse_Stop()***
- ***HAL_LPTIM_OnePulse_Start_IT()***
- ***HAL_LPTIM_OnePulse_Stop_IT()***

- *HAL_LPTIM_SetOnce_Start()*
- *HAL_LPTIM_SetOnce_Stop()*
- *HAL_LPTIM_SetOnce_Start_IT()*
- *HAL_LPTIM_SetOnce_Stop_IT()*
- *HAL_LPTIM_Encoder_Start()*
- *HAL_LPTIM_Encoder_Stop()*
- *HAL_LPTIM_Encoder_Start_IT()*
- *HAL_LPTIM_Encoder_Stop_IT()*
- *HAL_LPTIM_TimeOut_Start()*
- *HAL_LPTIM_TimeOut_Stop()*
- *HAL_LPTIM_TimeOut_Start_IT()*
- *HAL_LPTIM_TimeOut_Stop_IT()*
- *HAL_LPTIM_Counter_Start()*
- *HAL_LPTIM_Counter_Stop()*
- *HAL_LPTIM_Counter_Start_IT()*
- *HAL_LPTIM_Counter_Stop_IT()*

38.2.4 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare)value.

This section contains the following APIs:

- *HAL_LPTIM_ReadCounter()*
- *HAL_LPTIM_ReadAutoReload()*
- *HAL_LPTIM_ReadCompare()*

38.2.5 LPTIM IRQ handler and callbacks

This section provides LPTIM IRQ handler and callback functions called within the IRQ handler.

This section contains the following APIs:

- *HAL_LPTIM_IRQHandler()*
- *HAL_LPTIM_CompareMatchCallback()*
- *HAL_LPTIM_AutoReloadMatchCallback()*
- *HAL_LPTIM_TriggerCallback()*
- *HAL_LPTIM_CompareWriteCallback()*
- *HAL_LPTIM_AutoReloadWriteCallback()*
- *HAL_LPTIM_DirectionUpCallback()*
- *HAL_LPTIM_DirectionDownCallback()*

38.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_LPTIM_GetState()*

38.2.7 Detailed description of functions

HAL_LPTIM_Init

Function name	**HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hlptim)**
Function description	Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.
Parameters	• **hlptim:** LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_DeInit

Function name	**HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hlptim)**
Function description	DeInitialize the LPTIM peripheral.
Parameters	• **hlptim:** LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_MspInit

Function name	**void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)**
Function description	Initialize the LPTIM MSP.
Parameters	• **hlptim:** LPTIM handle
Return values	• **None:**

HAL_LPTIM_MspDeInit

Function name	**void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)**
Function description	DeInitialize LPTIM MSP.
Parameters	• **hlptim:** LPTIM handle
Return values	• **None:**

HAL_LPTIM_PWM_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Start the LPTIM PWM generation.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_PWM_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the LPTIM PWM generation.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_PWM_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Start the LPTIM PWM generation in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF
Return values	• **HAL:** status

HAL_LPTIM_PWM_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the LPTIM PWM generation in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_OnePulse_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Start the LPTIM One pulse generation.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_OnePulse_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the LPTIM One pulse generation.

Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_OnePulse_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Start the LPTIM One pulse generation in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_OnePulse_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the LPTIM One pulse generation in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_SetOnce_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Start the LPTIM in Set once mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_SetOnce_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the LPTIM Set once mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_SetOnce_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**
Function description	Start the LPTIM Set once mode in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_SetOnce_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the LPTIM Set once mode in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_Encoder_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)**
Function description	Start the Encoder interface.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_Encoder_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the Encoder interface.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_Encoder_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)**
Function description	Start the Encoder interface in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values	• **HAL:** status

HAL_LPTIM_Encoder_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the Encoder interface in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_TimeOut_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)**
Function description	Start the Timeout function.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Timeout:** : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status
Notes	• The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the Timeout function.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_TimeOut_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)**
Function description	Start the Timeout function in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • **Timeout:** : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

Notes	• The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the Timeout function in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_Counter_Start

Function name	**HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)**
Function description	Start the Counter mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_Counter_Stop

Function name	**HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the Counter mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **HAL:** status

HAL_LPTIM_Counter_Start_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)**
Function description	Start the Counter mode in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle • **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• **HAL:** status

HAL_LPTIM_Counter_Stop_IT

Function name	**HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hlptim)**
Function description	Stop the Counter mode in interrupt mode.
Parameters	• **hlptim:** : LPTIM handle

Return values	• **HAL:** status

HAL_LPTIM_ReadCounter

Function name	**uint32_t HAL_LPTIM_ReadCounter (LPTIM_HandleTypeDef * hlptim)**
Function description	Return the current counter value.
Parameters	• **hlptim:** LPTIM handle
Return values	• **Counter:** value.

HAL_LPTIM_ReadAutoReload

Function name	**uint32_t HAL_LPTIM_ReadAutoReload (LPTIM_HandleTypeDef * hlptim)**
Function description	Return the current Autoreload (Period) value.
Parameters	• **hlptim:** LPTIM handle
Return values	• **Autoreload:** value.

HAL_LPTIM_ReadCompare

Function name	**uint32_t HAL_LPTIM_ReadCompare (LPTIM_HandleTypeDef * hlptim)**
Function description	Return the current Compare (Pulse) value.
Parameters	• **hlptim:** LPTIM handle
Return values	• **Compare:** value.

HAL_LPTIM_IRQHandler

Function name	**void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)**
Function description	Handle LPTIM interrupt request.
Parameters	• **hlptim:** LPTIM handle
Return values	• **None:**

HAL_LPTIM_CompareMatchCallback

Function name	**void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Compare match callback in non-blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None:**

HAL_LPTIM_AutoReloadMatchCallback

Function name	**void HAL_LPTIM_AutoReloadMatchCallback**

	(LPTIM_HandleTypeDef * hlptim)
Function description	Autoreload match callback in non-blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None:**

HAL_LPTIM_TriggerCallback

Function name	**void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Trigger detected callback in non-blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None:**

HAL_LPTIM_CompareWriteCallback

Function name	**void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Compare write callback in non-blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None:**

HAL_LPTIM_AutoReloadWriteCallback

Function name	**void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Autoreload write callback in non-blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None:**

HAL_LPTIM_DirectionUpCallback

Function name	**void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Direction counter changed from Down to Up callback in non-blocking mode.
Parameters	• **hlptim:** : LPTIM handle
Return values	• **None:**

HAL_LPTIM_DirectionDownCallback

Function name	**void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hlptim)**
Function description	Direction counter changed from Up to Down callback in non-blocking mode.
Parameters	• **hlptim:** : LPTIM handle

Return values	• **None:**

HAL_LPTIM_GetState

Function name	**HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (LPTIM_HandleTypeDef * hlptim)**
Function description	Return the LPTIM handle state.
Parameters	• **hlptim:** LPTIM handle
Return values	• **HAL:** state

38.3 LPTIM Firmware driver defines

38.3.1 LPTIM

LPTIM Clock Polarity

LPTIM_CLOCKPOLARITY_RISING

LPTIM_CLOCKPOLARITY_FALLING

LPTIM_CLOCKPOLARITY_RISING_FALLING

LPTIM Clock Prescaler

LPTIM_PRESCALER_DIV1

LPTIM_PRESCALER_DIV2

LPTIM_PRESCALER_DIV4

LPTIM_PRESCALER_DIV8

LPTIM_PRESCALER_DIV16

LPTIM_PRESCALER_DIV32

LPTIM_PRESCALER_DIV64

LPTIM_PRESCALER_DIV128

LPTIM Clock Sample Time

LPTIM_CLOCKSAMPLETIME_DIRECTTRANSITION

LPTIM_CLOCKSAMPLETIME_2TRANSITIONS

LPTIM_CLOCKSAMPLETIME_4TRANSITIONS

LPTIM_CLOCKSAMPLETIME_8TRANSITIONS

LPTIM Clock Source

LPTIM_CLOCKSOURCE_APBCLOCK_LPOSC

LPTIM_CLOCKSOURCE_ULPTIM

LPTIM Counter Source

LPTIM_COUNTERSOURCE_INTERNAL

LPTIM_COUNTERSOURCE_EXTERNAL

LPTIM Exported Macros

__HAL_LPTIM_RESET_HANDLE_STATE

Description:

- Reset LPTIM handle state.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_ENABLE

Description:

- Enable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_DISABLE

Description:

- Disable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_START_CONTINUOUS

Description:

- Start the LPTIM peripheral in Continuous or in single mode.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_LPTIM_START_SINGLE

__HAL_LPTIM_AUTORELOAD_SET

Description:

- Write the passed parameter in the Autoreload register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Autoreload value

Return value:

- None

__HAL_LPTIM_COMPARE_SET

Description:

- Write the passed parameter in the

Compare register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Compare value

Return value:

- None

__HAL_LPTIM_GET_FLAG

Description:

- Check whether the specified LPTIM flag is set or not.

Parameters:

- __HANDLE__: LPTIM handle
- __FLAG__: LPTIM flag to check This parameter can be a value of:
  - LPTIM_FLAG_DOWN: Counter direction change up Flag.
  - LPTIM_FLAG_UP: Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK: Autoreload register update OK Flag.
  - LPTIM_FLAG_CMPOK: Compare register update OK Flag.
  - LPTIM_FLAG_EXTTRIG: External trigger edge event Flag.
  - LPTIM_FLAG_ARRM: Autoreload match Flag.
  - LPTIM_FLAG_CMPM: Compare match Flag.

Return value:

- The: state of the specified flag (SET or RESET).

__HAL_LPTIM_CLEAR_FLAG

Description:

- Clear the specified LPTIM flag.

Parameters:

- __HANDLE__: LPTIM handle.
- __FLAG__: LPTIM flag to clear. This parameter can be a value of:
  - LPTIM_FLAG_DOWN: Counter direction change up Flag.
  - LPTIM_FLAG_UP: Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK: Autoreload register update OK Flag.
  - LPTIM_FLAG_CMPOK: Compare register update OK Flag.
  - LPTIM_FLAG_EXTTRIG: External trigger edge event Flag.
  - LPTIM_FLAG_ARRM: Autoreload

match Flag.
  - LPTIM_FLAG_CMPM: Compare match Flag.

Return value:

- None

__HAL_LPTIM_ENABLE_IT

Description:

- Enable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_DOWN: Counter direction change up Interrupt.
  - LPTIM_IT_UP: Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK: Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK: Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG: External trigger edge event Interrupt.
  - LPTIM_IT_ARRM: Autoreload match Interrupt.
  - LPTIM_IT_CMPM: Compare match Interrupt.

Return value:

- None

__HAL_LPTIM_DISABLE_IT

Description:

- Disable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_DOWN: Counter direction change up Interrupt.
  - LPTIM_IT_UP: Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK: Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK: Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG: External trigger edge event Interrupt.
  - LPTIM_IT_ARRM: Autoreload match Interrupt.
  - LPTIM_IT_CMPM: Compare match Interrupt.

Return value:

- None

__HAL_LPTIM_GET_IT_SOURCE

Description:

- Check whether the specified LPTIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to check. This parameter can be a value of:
  - LPTIM_IT_DOWN: Counter direction change up Interrupt.
  - LPTIM_IT_UP: Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK: Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK: Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG: External trigger edge event Interrupt.
  - LPTIM_IT_ARRM: Autoreload match Interrupt.
  - LPTIM_IT_CMPM: Compare match Interrupt.

Return value:

- Interrupt: status.

LPTIM External Trigger Polarity

LPTIM_ACTIVEEDGE_RISING

LPTIM_ACTIVEEDGE_FALLING

LPTIM_ACTIVEEDGE_RISING_FALLING

LPTIM Flags Definition

LPTIM_FLAG_DOWN

LPTIM_FLAG_UP

LPTIM_FLAG_ARROK

LPTIM_FLAG_CMPOK

LPTIM_FLAG_EXTTRIG

LPTIM_FLAG_ARRM

LPTIM_FLAG_CMPM

LPTIM Input1 Source

LPTIM_INPUT1SOURCE_GPIO	For LPTIM1 and LPTIM2
LPTIM_INPUT1SOURCE_COMP1	For LPTIM1 and LPTIM2
LPTIM_INPUT1SOURCE_COMP2	For LPTIM2
LPTIM_INPUT1SOURCE_COMP1_COMP2	For LPTIM2

LPTIM Input2 Source

LPTIM_INPUT2SOURCE_GPIO	For LPTIM1
LPTIM_INPUT2SOURCE_COMP2	For LPTIM1

LPTIM Interrupts Definition

LPTIM_IT_DOWN

LPTIM_IT_UP

LPTIM_IT_ARROK

LPTIM_IT_CMPOK

LPTIM_IT_EXTTRIG

LPTIM_IT_ARRM

LPTIM_IT_CMPM

LPTIM Output Polarity

LPTIM_OUTPUTPOLARITY_HIGH

LPTIM_OUTPUTPOLARITY_LOW

LPTIM Trigger Sample Time

LPTIM_TRIGSAMPLETIME_DIRECTTRANSITION

LPTIM_TRIGSAMPLETIME_2TRANSITIONS

LPTIM_TRIGSAMPLETIME_4TRANSITIONS

LPTIM_TRIGSAMPLETIME_8TRANSITIONS

LPTIM Trigger Source

LPTIM_TRIGSOURCE_SOFTWARE

LPTIM_TRIGSOURCE_0

LPTIM_TRIGSOURCE_1

LPTIM_TRIGSOURCE_2

LPTIM_TRIGSOURCE_3

LPTIM_TRIGSOURCE_4

LPTIM_TRIGSOURCE_5

LPTIM_TRIGSOURCE_6

LPTIM_TRIGSOURCE_7

LPTIM Updating Mode

LPTIM_UPDATE_IMMEDIATE

LPTIM_UPDATE_ENDOFPERIOD

39 HAL LTDC Generic Driver

39.1 LTDC Firmware driver registers structures

39.1.1 LTDC_ColorTypeDef

Data Fields

- ***uint8_t Blue***
- ***uint8_t Green***
- ***uint8_t Red***
- ***uint8_t Reserved***

Field Documentation

- ***uint8_t LTDC_ColorTypeDef::Blue***
  Configures the blue value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Green***
  Configures the green value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Red***
  Configures the red value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Reserved***
  Reserved 0xFF

39.1.2 LTDC_InitTypeDef

Data Fields

- ***uint32_t HSPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t DEPolarity***
- ***uint32_t PCPolarity***
- ***uint32_t HorizontalSync***
- ***uint32_t VerticalSync***
- ***uint32_t AccumulatedHBP***
- ***uint32_t AccumulatedVBP***
- ***uint32_t AccumulatedActiveW***
- ***uint32_t AccumulatedActiveH***
- ***uint32_t TotalWidth***
- ***uint32_t TotalHeigh***
- ***LTDC_ColorTypeDef Backcolor***

Field Documentation

- ***uint32_t LTDC_InitTypeDef::HSPolarity***
  configures the horizontal synchronization polarity. This parameter can be one value of ***LTDC_HS_POLARITY***
- ***uint32_t LTDC_InitTypeDef::VSPolarity***
  configures the vertical synchronization polarity. This parameter can be one value of ***LTDC_VS_POLARITY***

- ***uint32_t LTDC_InitTypeDef::DEPolarity***
  configures the data enable polarity. This parameter can be one of value of ***LTDC_DE_POLARITY***
- ***uint32_t LTDC_InitTypeDef::PCPolarity***
  configures the pixel clock polarity. This parameter can be one of value of ***LTDC_PC_POLARITY***
- ***uint32_t LTDC_InitTypeDef::HorizontalSync***
  configures the number of Horizontal synchronization width. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::VerticalSync***
  configures the number of Vertical synchronization height. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedHBP***
  configures the accumulated horizontal back porch width. This parameter must be a number between Min_Data = LTDC_HorizontalSync and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedVBP***
  configures the accumulated vertical back porch height. This parameter must be a number between Min_Data = LTDC_VerticalSync and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedActiveW***
  configures the accumulated active width. This parameter must be a number between Min_Data = LTDC_AccumulatedHBP and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedActiveH***
  configures the accumulated active height. This parameter must be a number between Min_Data = LTDC_AccumulatedVBP and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::TotalWidth***
  configures the total width. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveW and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::TotalHeigh***
  configures the total height. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveH and Max_Data = 0x7FF.
- ***LTDC_ColorTypeDef LTDC_InitTypeDef::Backcolor***
  Configures the background color.

39.1.3 LTDC_LayerCfgTypeDef

Data Fields

- ***uint32_t WindowX0***
- ***uint32_t WindowX1***
- ***uint32_t WindowY0***
- ***uint32_t WindowY1***
- ***uint32_t PixelFormat***
- ***uint32_t Alpha***
- ***uint32_t Alpha0***
- ***uint32_t BlendingFactor1***
- ***uint32_t BlendingFactor2***
- ***uint32_t FBStartAdress***
- ***uint32_t ImageWidth***
- ***uint32_t ImageHeight***
- ***LTDC_ColorTypeDef Backcolor***

Field Documentation

- ***uint32_t LTDC_LayerCfgTypeDef::WindowX0***
  Configures the Window Horizontal Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.

- ***uint32_t LTDC_LayerCfgTypeDef::WindowX1***
  Configures the Window Horizontal Stop Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_LayerCfgTypeDef::WindowY0***
  Configures the Window vertical Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_LayerCfgTypeDef::WindowY1***
  Configures the Window vertical Stop Position. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_LayerCfgTypeDef::PixelFormat***
  Specifies the pixel format. This parameter can be one of value of ***LTDC_Pixelformat***
- ***uint32_t LTDC_LayerCfgTypeDef::Alpha***
  Specifies the constant alpha used for blending. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t LTDC_LayerCfgTypeDef::Alpha0***
  Configures the default alpha value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t LTDC_LayerCfgTypeDef::BlendingFactor1***
  Select the blending factor 1. This parameter can be one of value of ***LTDC_BlendingFactor1***
- ***uint32_t LTDC_LayerCfgTypeDef::BlendingFactor2***
  Select the blending factor 2. This parameter can be one of value of ***LTDC_BlendingFactor2***
- ***uint32_t LTDC_LayerCfgTypeDef::FBStartAdress***
  Configures the color frame buffer address
- ***uint32_t LTDC_LayerCfgTypeDef::ImageWidth***
  Configures the color frame buffer line length. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x1FFF.
- ***uint32_t LTDC_LayerCfgTypeDef::ImageHeight***
  Specifies the number of line in frame buffer. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***LTDC_ColorTypeDef LTDC_LayerCfgTypeDef::Backcolor***
  Configures the layer background color.

39.1.4 LTDC_HandleTypeDef

Data Fields

- ***LTDC_TypeDef * Instance***
- ***LTDC_InitTypeDef Init***
- ***LTDC_LayerCfgTypeDef LayerCfg***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LTDC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***LTDC_TypeDef* LTDC_HandleTypeDef::Instance***
  LTDC Register base address
- ***LTDC_InitTypeDef LTDC_HandleTypeDef::Init***
  LTDC parameters
- ***LTDC_LayerCfgTypeDef LTDC_HandleTypeDef::LayerCfg[MAX_LAYER]***
  LTDC Layers parameters
- ***HAL_LockTypeDef LTDC_HandleTypeDef::Lock***
  LTDC Lock

- ***__IO HAL_LTDC_StateTypeDef LTDC_HandleTypeDef::State***
  LTDC state
- ***__IO uint32_t LTDC_HandleTypeDef::ErrorCode***
  LTDC Error code

39.2 LTDC Firmware driver API description

39.2.1 How to use this driver

1. Program the required configuration through the following parameters: the LTDC timing, the horizontal and vertical polarity, the pixel clock polarity, Data Enable polarity and the LTDC background color value using HAL_LTDC_Init() function
2. Program the required configuration through the following parameters: the pixel format, the blending factors, input alpha value, the window size and the image size using HAL_LTDC_ConfigLayer() function for foreground or/and background layer.
3. Optionally, configure and enable the CLUT using HAL_LTDC_ConfigCLUT() and HAL_LTDC_EnableCLUT functions.
4. Optionally, enable the Dither using HAL_LTDC_EnableDither().
5. Optionally, configure and enable the Color keying using HAL_LTDC_ConfigColorKeying() and HAL_LTDC_EnableColorKeying functions.
6. Optionally, configure LineInterrupt using HAL_LTDC_ProgramLineEvent() function
7. If needed, reconfigure and change the pixel format value, the alpha value value, the window size, the window position and the layer start address for foreground or/and background layer using respectively the following functions: HAL_LTDC_SetPixelFormat(), HAL_LTDC_SetAlpha(), HAL_LTDC_SetWindowSize(), HAL_LTDC_SetWindowPosition() and HAL_LTDC_SetAddress().
8. Variant functions with _NoReload suffix allows to set the LTDC configuration/settings without immediate reload. This is useful in case when the program requires to modify serval LTDC settings (on one or both layers) then applying(reload) these settings in one shot by calling the function HAL_LTDC_Reload(). After calling the _NoReload functions to set different color/format/layer settings, the program shall call the function HAL_LTDC_Reload() to apply(reload) these settings. Function HAL_LTDC_Reload() can be called with the parameter ReloadType set to LTDC_RELOAD_IMMEDIATE if an immediate reload is required. Function HAL_LTDC_Reload() can be called with the parameter ReloadType set to LTDC_RELOAD_VERTICAL_BLANKING if the reload should be done in the next vertical blanking period, this option allows to avoid display flicker by applying the new settings during the vertical blanking period.
9. To control LTDC state you can use the following function: HAL_LTDC_GetState()

LTDC HAL driver macros list

Below the list of most used macros in LTDC HAL driver.

- __HAL_LTDC_ENABLE: Enable the LTDC.
- __HAL_LTDC_DISABLE: Disable the LTDC.
- __HAL_LTDC_LAYER_ENABLE: Enable a LTDC Layer.
- __HAL_LTDC_LAYER_DISABLE: Disable a LTDC Layer.
- __HAL_LTDC_CLEAR_FLAG: Clear the LTDC pending flags.
- __HAL_LTDC_ENABLE_IT: Enable the specified LTDC interrupts.
- __HAL_LTDC_DISABLE_IT: Disable the specified LTDC interrupts.

You can refer to the LTDC HAL driver header file for more useful macros

39.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC
- De-initialize the LTDC

This section contains the following APIs:

- ***HAL_LTDC_Init()***
- ***HAL_LTDC_DeInit()***
- ***HAL_LTDC_MspInit()***
- ***HAL_LTDC_MspDeInit()***
- ***HAL_LTDC_ErrorCallback()***
- ***HAL_LTDC_LineEventCallback()***
- ***HAL_LTDC_ReloadEventCallback()***

39.2.3 IO operation functions

This section provides function allowing to:

- Handle LTDC interrupt request

This section contains the following APIs:

- ***HAL_LTDC_IRQHandler()***
- ***HAL_LTDC_ErrorCallback()***
- ***HAL_LTDC_LineEventCallback()***
- ***HAL_LTDC_ReloadEventCallback()***

39.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the LTDC foreground or/and background parameters.
- Set the active layer.
- Configure the color keying.
- Configure the C-LUT.
- Enable / Disable the color keying.
- Enable / Disable the C-LUT.
- Update the layer position.
- Update the layer size.
- Update pixel format on the fly.
- Update transparency on the fly.
- Update address on the fly.

This section contains the following APIs:

- ***HAL_LTDC_ConfigLayer()***
- ***HAL_LTDC_ConfigColorKeying()***
- ***HAL_LTDC_ConfigCLUT()***
- ***HAL_LTDC_EnableColorKeying()***
- ***HAL_LTDC_DisableColorKeying()***
- ***HAL_LTDC_EnableCLUT()***

- ***HAL_LTDC_DisableCLUT()***
- ***HAL_LTDC_EnableDither()***
- ***HAL_LTDC_DisableDither()***
- ***HAL_LTDC_SetWindowSize()***
- ***HAL_LTDC_SetWindowPosition()***
- ***HAL_LTDC_SetPixelFormat()***
- ***HAL_LTDC_SetAlpha()***
- ***HAL_LTDC_SetAddress()***
- ***HAL_LTDC_SetPitch()***
- ***HAL_LTDC_ProgramLineEvent()***
- ***HAL_LTDC_Reload()***
- ***HAL_LTDC_ConfigLayer_NoReload()***
- ***HAL_LTDC_SetWindowSize_NoReload()***
- ***HAL_LTDC_SetWindowPosition_NoReload()***
- ***HAL_LTDC_SetPixelFormat_NoReload()***
- ***HAL_LTDC_SetAlpha_NoReload()***
- ***HAL_LTDC_SetAddress_NoReload()***
- ***HAL_LTDC_SetPitch_NoReload()***
- ***HAL_LTDC_ConfigColorKeying_NoReload()***
- ***HAL_LTDC_EnableColorKeying_NoReload()***
- ***HAL_LTDC_DisableColorKeying_NoReload()***
- ***HAL_LTDC_EnableCLUT_NoReload()***
- ***HAL_LTDC_DisableCLUT_NoReload()***

39.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the LTDC handle state.
- Get the LTDC handle error code.

This section contains the following APIs:

- ***HAL_LTDC_GetState()***
- ***HAL_LTDC_GetError()***

39.2.6 Detailed description of functions

HAL_LTDC_Init

Function name	**HAL_StatusTypeDef HAL_LTDC_Init (LTDC_HandleTypeDef * hltdc)**
Function description	Initialize the LTDC according to the specified parameters in the LTDC_InitTypeDef.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **HAL:** status

HAL_LTDC_DeInit

Function name	**HAL_StatusTypeDef HAL_LTDC_DeInit (LTDC_HandleTypeDef * hltdc)**
Function description	De-initialize the LTDC peripheral.

Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **None:**

HAL_LTDC_MspInit

Function name	**void HAL_LTDC_MspInit (LTDC_HandleTypeDef * hltdc)**
Function description	Initialize the LTDC MSP.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **None:**

HAL_LTDC_MspDeInit

Function name	**void HAL_LTDC_MspDeInit (LTDC_HandleTypeDef * hltdc)**
Function description	De-initialize the LTDC MSP.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **None:**

HAL_LTDC_ErrorCallback

Function name	**void HAL_LTDC_ErrorCallback (LTDC_HandleTypeDef * hltdc)**
Function description	Error LTDC callback.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **None:**

HAL_LTDC_LineEventCallback

Function name	**void HAL_LTDC_LineEventCallback (LTDC_HandleTypeDef * hltdc)**
Function description	Line Event callback.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **None:**

HAL_LTDC_ReloadEventCallback

Function name	**void HAL_LTDC_ReloadEventCallback (LTDC_HandleTypeDef * hltdc)**
Function description	Reload Event callback.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **None:**

HAL_LTDC_IRQHandler

Function name
: **void HAL_LTDC_IRQHandler (LTDC_HandleTypeDef * hltdc)**

Function description
: Handle LTDC interrupt request.

Parameters
: - **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values
: - **HAL:** status

HAL_LTDC_ConfigLayer

Function name
: **HAL_StatusTypeDef HAL_LTDC_ConfigLayer (LTDC_HandleTypeDef * hltdc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)**

Function description
: Configure the LTDC Layer according to the specified parameters in the LTDC_InitTypeDef and create the associated handle.

Parameters
: - **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
  - **pLayerCfg:** pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer.
  - **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values
: - **HAL:** status

HAL_LTDC_SetWindowSize

Function name
: **HAL_StatusTypeDef HAL_LTDC_SetWindowSize (LTDC_HandleTypeDef * hltdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)**

Function description
: Set the LTDC window size.

Parameters
: - **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
  - **XSize:** LTDC Pixel per line
  - **YSize:** LTDC Line number
  - **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values
: - **HAL:** status

HAL_LTDC_SetWindowPosition

Function name
: **HAL_StatusTypeDef HAL_LTDC_SetWindowPosition (LTDC_HandleTypeDef * hltdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)**

Function description
: Set the LTDC window position.

Parameters
: - **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
  - **X0:** LTDC window X offset
  - **Y0:** LTDC window Y offset

	• **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_SetPixelFormat

Function name	**HAL_StatusTypeDef HAL_LTDC_SetPixelFormat (LTDC_HandleTypeDef * hltdc, uint32_t Pixelformat, uint32_t LayerIdx)**
Function description	Reconfigure the pixel format.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **Pixelformat:** new pixel format value. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).
Return values	• **HAL:** status

HAL_LTDC_SetAlpha

Function name	**HAL_StatusTypeDef HAL_LTDC_SetAlpha (LTDC_HandleTypeDef * hltdc, uint32_t Alpha, uint32_t LayerIdx)**
Function description	Reconfigure the layer alpha value.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **Alpha:** new alpha value. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_SetAddress

Function name	**HAL_StatusTypeDef HAL_LTDC_SetAddress (LTDC_HandleTypeDef * hltdc, uint32_t Address, uint32_t LayerIdx)**
Function description	Reconfigure the frame buffer Address.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **Address:** new address value. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).
Return values	• **HAL:** status

HAL_LTDC_SetPitch

Function name	**HAL_StatusTypeDef HAL_LTDC_SetPitch (LTDC_HandleTypeDef * hltdc, uint32_t LinePitchInPixels, uint32_t LayerIdx)**
Function description	Function used to reconfigure the pitch for specific cases where the attached LayerIdx buffer have a width that is larger than the one intended to be displayed on screen.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **LinePitchInPixels:** New line pitch in pixels to configure for LTDC layer 'LayerIdx'. • **LayerIdx:** LTDC layer index concerned by the modification of line pitch.
Return values	• **HAL:** status
Notes	• This function should be called only after a previous call to HAL_LTDC_ConfigLayer() to modify the default pitch configured by HAL_LTDC_ConfigLayer() when required (refer to example described just above).

HAL_LTDC_ConfigColorKeying

Function name	**HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t RGBValue, uint32_t LayerIdx)**
Function description	Configure the color keying.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **RGBValue:** the color key value • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_ConfigCLUT

Function name	**HAL_StatusTypeDef HAL_LTDC_ConfigCLUT (LTDC_HandleTypeDef * hltdc, uint32_t * pCLUT, uint32_t CLUTSize, uint32_t LayerIdx)**
Function description	Load the color lookup table.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **pCLUT:** pointer to the color lookup table address. • **CLUTSize:** the color lookup table size. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_EnableColorKeying

Function name	**HAL_StatusTypeDef HAL_LTDC_EnableColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)**
Function description	Enable the color keying.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_DisableColorKeying

Function name	**HAL_StatusTypeDef HAL_LTDC_DisableColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)**
Function description	Disable the color keying.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_EnableCLUT

Function name	**HAL_StatusTypeDef HAL_LTDC_EnableCLUT (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)**
Function description	Enable the color lookup table.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_DisableCLUT

Function name	**HAL_StatusTypeDef HAL_LTDC_DisableCLUT (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)**
Function description	Disable the color lookup table.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_ProgramLineEvent

Function name	**HAL_StatusTypeDef HAL_LTDC_ProgramLineEvent (LTDC_HandleTypeDef * hltdc, uint32_t Line)**
Function description	Define the position of the line interrupt.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **Line:** Line Interrupt Position.
Return values	• **HAL:** status
Notes	• User application may resort to HAL_LTDC_LineEventCallback() at line interrupt generation.

HAL_LTDC_EnableDither

Function name	**HAL_StatusTypeDef HAL_LTDC_EnableDither (LTDC_HandleTypeDef * hltdc)**
Function description	Enable Dither.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **HAL:** status

HAL_LTDC_DisableDither

Function name	**HAL_StatusTypeDef HAL_LTDC_DisableDither (LTDC_HandleTypeDef * hltdc)**
Function description	Disable Dither.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **HAL:** status

HAL_LTDC_Reload

Function name	**HAL_StatusTypeDef HAL_LTDC_Reload (LTDC_HandleTypeDef * hltdc, uint32_t ReloadType)**
Function description	Reload LTDC Layers configuration.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **ReloadType:** This parameter can be one of the following values: LTDC_RELOAD_IMMEDIATE: Immediate Reload LTDC_RELOAD_VERTICAL_BLANKING: Reload in the next Vertical Blanking
Return values	• **HAL:** status
Notes	• User application may resort to HAL_LTDC_ReloadEventCallback() at reload interrupt generation.

HAL_LTDC_ConfigLayer_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_ConfigLayer_NoReload (LTDC_HandleTypeDef * hltdc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)**
Function description	Configure the LTDC Layer according to the specified without reloading parameters in the LTDC_InitTypeDef and create the associated handle.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **pLayerCfg:** pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_SetWindowSize_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_SetWindowSize_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)**
Function description	Set the LTDC window size without reloading.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **XSize:** LTDC Pixel per line • **YSize:** LTDC Line number • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_SetWindowPosition_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_SetWindowPosition_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)**
Function description	Set the LTDC window position without reloading.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **X0:** LTDC window X offset • **Y0:** LTDC window Y offset • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_SetPixelFormat_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_SetPixelFormat_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Pixelformat, uint32_t LayerIdx)**
Function description	Reconfigure the pixel format without reloading.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDfef structure that contains the configuration information for the LTDC. • **Pixelformat:** new pixel format value. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).
Return values	• **HAL:** status

HAL_LTDC_SetAlpha_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_SetAlpha_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Alpha, uint32_t LayerIdx)**
Function description	Reconfigure the layer alpha value without reloading.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **Alpha:** new alpha value. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_SetAddress_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_SetAddress_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Address, uint32_t LayerIdx)**
Function description	Reconfigure the frame buffer Address without reloading.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **Address:** new address value. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).
Return values	• **HAL:** status

HAL_LTDC_SetPitch_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_SetPitch_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LinePitchInPixels, uint32_t LayerIdx)**
Function description	Function used to reconfigure the pitch for specific cases where the attached LayerIdx buffer have a width that is larger than the one

intended to be displayed on screen.

Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **LinePitchInPixels:** New line pitch in pixels to configure for LTDC layer 'LayerIdx'. • **LayerIdx:** LTDC layer index concerned by the modification of line pitch.
Return values	• **HAL:** status
Notes	• This function should be called only after a previous call to HAL_LTDC_ConfigLayer() to modify the default pitch configured by HAL_LTDC_ConfigLayer() when required (refer to example described just above). Variant of the function HAL_LTDC_SetPitch without immediate reload.

HAL_LTDC_ConfigColorKeying_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t RGBValue, uint32_t LayerIdx)**
Function description	Configure the color keying without reloading.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **RGBValue:** the color key value • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_EnableColorKeying_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_EnableColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)**
Function description	Enable the color keying without reloading.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_DisableColorKeying_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_DisableColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)**
Function description	Disable the color keying without reloading.

Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_EnableCLUT_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_EnableCLUT_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)**
Function description	Enable the color lookup table without reloading.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_DisableCLUT_NoReload

Function name	**HAL_StatusTypeDef HAL_LTDC_DisableCLUT_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)**
Function description	Disable the color lookup table without reloading.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)
Return values	• **HAL:** status

HAL_LTDC_GetState

Function name	**HAL_LTDC_StateTypeDef HAL_LTDC_GetState (LTDC_HandleTypeDef * hltdc)**
Function description	Return the LTDC handle state.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **HAL:** state

HAL_LTDC_GetError

Function name	**uint32_t HAL_LTDC_GetError (LTDC_HandleTypeDef * hltdc)**
Function description	Return the LTDC handle error code.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **LTDC:** Error Code

39.3 LTDC Firmware driver defines

39.3.1 LTDC

LTDC Alpha

LTDC_ALPHA	LTDC Constant Alpha mask

LTDC BACK COLOR

LTDC_COLOR	Color mask

LTDC Blending Factor1

LTDC_BLENDING_FACTOR1_CA	Blending factor: Cte Alpha
LTDC_BLENDING_FACTOR1_PAxCA	Blending factor: Cte Alpha x Pixel Alpha

LTDC Blending Factor2

LTDC_BLENDING_FACTOR2_CA	Blending factor: Cte Alpha
LTDC_BLENDING_FACTOR2_PAxCA	Blending factor: Cte Alpha x Pixel Alpha

LTDC DE POLARITY

LTDC_DEPOLARITY_AL	Data Enable, is active low.
LTDC_DEPOLARITY_AH	Data Enable, is active high.

LTDC Error Code

HAL_LTDC_ERROR_NONE	LTDC No error
HAL_LTDC_ERROR_TE	LTDC Transfer error
HAL_LTDC_ERROR_FU	LTDC FIFO Underrun
HAL_LTDC_ERROR_TIMEOUT	LTDC Timeout error

LTDC Exported Macros

__HAL_LTDC_RESET_HANDLE_STATE

Description:

- Reset LTDC handle state.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None

__HAL_LTDC_ENABLE

Description:

- Enable the LTDC.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_DISABLE

Description:

- Disable the LTDC.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_LAYER_ENABLE

Description:

- Enable the LTDC Layer.

Parameters:

- __HANDLE__: LTDC handle
- __LAYER__: Specify the layer to be enabled. This parameter can be LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return value:

- None.

__HAL_LTDC_LAYER_DISABLE

Description:

- Disable the LTDC Layer.

Parameters:

- __HANDLE__: LTDC handle
- __LAYER__: Specify the layer to be disabled. This parameter can be LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return value:

- None.

__HAL_LTDC_RELOAD_IMMEDIATE_CONFIG

Description:

- Reload immediately all LTDC Layers.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_VERTICAL_BLANKING_RELOAD_CONFIG

Description:

- Reload during vertical blanking period all LTDC Layers.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_GET_FLAG

Description:

- Get the LTDC pending flags.

Parameters:

- __HANDLE__: LTDC handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - LTDC_FLAG_LI: Line Interrupt flag
  - LTDC_FLAG_FU: FIFO Underrun Interrupt flag
  - LTDC_FLAG_TE: Transfer Error interrupt flag
  - LTDC_FLAG_RR: Register Reload Interrupt Flag

Return value:

- The: state of FLAG (SET or RESET).

__HAL_LTDC_CLEAR_FLAG

Description:

- Clears the LTDC pending flags.

Parameters:

- __HANDLE__: LTDC handle
- __FLAG__: Specify the flag to clear. This parameter can be any combination of the following values:
  - LTDC_FLAG_LI: Line Interrupt flag
  - LTDC_FLAG_FU: FIFO Underrun Interrupt flag
  - LTDC_FLAG_TE: Transfer Error interrupt flag
  - LTDC_FLAG_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_ENABLE_IT

Description:

- Enables the specified LTDC interrupts.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - LTDC_IT_LI: Line Interrupt

flag
- LTDC_IT_FU: FIFO Underrun Interrupt flag
- LTDC_IT_TE: Transfer Error interrupt flag
- LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_DISABLE_IT

Description:

- Disables the specified LTDC interrupts.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - LTDC_IT_LI: Line Interrupt flag
  - LTDC_IT_FU: FIFO Underrun Interrupt flag
  - LTDC_IT_TE: Transfer Error interrupt flag
  - LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_GET_IT_SOURCE

Description:

- Check whether the specified LTDC interrupt has occurred or not.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt source to check. This parameter can be one of the following values:
  - LTDC_IT_LI: Line Interrupt flag
  - LTDC_IT_FU: FIFO Underrun Interrupt flag
  - LTDC_IT_TE: Transfer Error interrupt flag
  - LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- The: state of INTERRUPT (SET or RESET).

LTDC Exported Types

MAX_LAYER

LTDC Flags

LTDC_FLAG_LI	LTDC Line Interrupt Flag
LTDC_FLAG_FU	LTDC FIFO Underrun interrupt Flag
LTDC_FLAG_TE	LTDC Transfer Error interrupt Flag
LTDC_FLAG_RR	LTDC Register Reload interrupt Flag

LTDC HS POLARITY

LTDC_HSPOLARITY_AL	Horizontal Synchronization is active low.
LTDC_HSPOLARITY_AH	Horizontal Synchronization is active high.

LTDC Interrupts

LTDC_IT_LI	LTDC Line Interrupt
LTDC_IT_FU	LTDC FIFO Underrun Interrupt
LTDC_IT_TE	LTDC Transfer Error Interrupt
LTDC_IT_RR	LTDC Register Reload Interrupt

LTDC Layer

LTDC_LAYER_1	LTDC Layer 1
LTDC_LAYER_2	LTDC Layer 2

LTDC LAYER Config

LTDC_STOPPOSITION	LTDC Layer stop position
LTDC_STARTPOSITION	LTDC Layer start position
LTDC_COLOR_FRAME_BUFFER	LTDC Layer Line length
LTDC_LINE_NUMBER	LTDC Layer Line number

LTDC PC POLARITY

LTDC_PCPOLARITY_IPC	input pixel clock.
LTDC_PCPOLARITY_IIPC	inverted input pixel clock.

LTDC Pixel format

LTDC_PIXEL_FORMAT_ARGB8888	ARGB8888 LTDC pixel format
LTDC_PIXEL_FORMAT_RGB888	RGB888 LTDC pixel format
LTDC_PIXEL_FORMAT_RGB565	RGB565 LTDC pixel format
LTDC_PIXEL_FORMAT_ARGB1555	ARGB1555 LTDC pixel format
LTDC_PIXEL_FORMAT_ARGB4444	ARGB4444 LTDC pixel format
LTDC_PIXEL_FORMAT_L8	L8 LTDC pixel format
LTDC_PIXEL_FORMAT_AL44	AL44 LTDC pixel format

LTDC_PIXEL_FORMAT_AL88	AL88 LTDC pixel format

LTDC Reload Type

LTDC_RELOAD_IMMEDIATE	Immediate Reload
LTDC_RELOAD_VERTICAL_BLANKING	Vertical Blanking Reload

LTDC SYNC

LTDC_HORIZONTALSYNC	Horizontal synchronization width.
LTDC_VERTICALSYNC	Vertical synchronization height.

LTDC VS POLARITY

LTDC_VSPOLARITY_AL	Vertical Synchronization is active low.
LTDC_VSPOLARITY_AH	Vertical Synchronization is active high.

40 HAL LTDC Extension Driver

40.1 LTDCEx Firmware driver API description

40.1.1 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC

This section contains the following APIs:

- *HAL_LTDCEx_StructInitFromVideoConfig()*
- *HAL_LTDCEx_StructInitFromAdaptedCommandConfig()*

40.1.2 Detailed description of functions

HAL_LTDCEx_StructInitFromVideoConfig

Function name	**HAL_StatusTypeDef HAL_LTDCEx_StructInitFromVideoConfig (LTDC_HandleTypeDef * hltdc, DSI_VidCfgTypeDef * VidCfg)**
Function description	Retrieve common parameters from DSI Video mode configuration structure.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **VidCfg:** pointer to a DSI_VidCfgTypeDef structure that contains the DSI video mode configuration parameters
Return values	• **HAL:** status
Notes	• The implementation of this function is taking into account the LTDC polarities inversion as described in the current LTDC specification

HAL_LTDCEx_StructInitFromAdaptedCommandConfig

Function name	**HAL_StatusTypeDef HAL_LTDCEx_StructInitFromAdaptedCommandConfig (LTDC_HandleTypeDef * hltdc, DSI_CmdCfgTypeDef * CmdCfg)**
Function description	Retrieve common parameters from DSI Adapted command mode configuration structure.
Parameters	• **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • **CmdCfg:** pointer to a DSI_CmdCfgTypeDef structure that contains the DSI command mode configuration parameters
Return values	• **HAL:** status
Notes	• The implementation of this function is taking into account the LTDC polarities inversion as described in the current LTDC specification

41 HAL NAND Generic Driver

41.1 NAND Firmware driver registers structures

41.1.1 NAND_IDTypeDef

Data Fields

- ***uint8_t Maker_Id***
- ***uint8_t Device_Id***
- ***uint8_t Third_Id***
- ***uint8_t Fourth_Id***

Field Documentation

- ***uint8_t NAND_IDTypeDef::Maker_Id***
- ***uint8_t NAND_IDTypeDef::Device_Id***
- ***uint8_t NAND_IDTypeDef::Third_Id***
- ***uint8_t NAND_IDTypeDef::Fourth_Id***

41.1.2 NAND_AddressTypeDef

Data Fields

- ***uint16_t Page***
- ***uint16_t Zone***
- ***uint16_t Block***

Field Documentation

- ***uint16_t NAND_AddressTypeDef::Page***
  NAND memory Page address
- ***uint16_t NAND_AddressTypeDef::Zone***
  NAND memory Zone address
- ***uint16_t NAND_AddressTypeDef::Block***
  NAND memory Block address

41.1.3 NAND_InfoTypeDef

Data Fields

- ***uint32_t PageSize***
- ***uint32_t SpareAreaSize***
- ***uint32_t BlockSize***
- ***uint32_t BlockNbr***
- ***uint32_t ZoneSize***

Field Documentation

- ***uint32_t NAND_InfoTypeDef::PageSize***
  NAND memory page (without spare area) size measured in K. bytes
- ***uint32_t NAND_InfoTypeDef::SpareAreaSize***
  NAND memory spare area size measured in K. bytes
- ***uint32_t NAND_InfoTypeDef::BlockSize***
  NAND memory block size number of pages
- ***uint32_t NAND_InfoTypeDef::BlockNbr***
  NAND memory number of blocks

- ***uint32_t NAND_InfoTypeDef::ZoneSize***
  NAND memory zone size measured in number of blocks

41.1.4 NAND_HandleTypeDef

Data Fields

- ***FMC_NAND_TypeDef * Instance***
- ***FMC_NAND_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NAND_StateTypeDef State***
- ***NAND_InfoTypeDef Info***

Field Documentation

- ***FMC_NAND_TypeDef* NAND_HandleTypeDef::Instance***
  Register base address
- ***FMC_NAND_InitTypeDef NAND_HandleTypeDef::Init***
  NAND device control configuration parameters
- ***HAL_LockTypeDef NAND_HandleTypeDef::Lock***
  NAND locking object
- ***__IO HAL_NAND_StateTypeDef NAND_HandleTypeDef::State***
  NAND device access state
- ***NAND_InfoTypeDef NAND_HandleTypeDef::Info***
  NAND characteristic information structure

41.2 NAND Firmware driver API description

41.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function HAL_NAND_Init() with control and timing parameters for both common and attribute spaces.
- Read NAND flash memory maker and device IDs using the function HAL_NAND_Read_ID(). The read information is stored in the NAND_ID_TypeDef structure declared by the function caller.
- Access NAND flash memory by read/write operations using the functions HAL_NAND_Read_Page()/HAL_NAND_Read_SpareArea(), HAL_NAND_Write_Page()/HAL_NAND_Write_SpareArea() to read/write page(s)/spare area(s). These functions use specific device information (Block, page size..) predefined by the user in the HAL_NAND_Info_TypeDef structure. The read/write address information is contained by the Nand_Address_Typedef structure passed as parameter.
- Perform NAND flash Reset chip operation using the function HAL_NAND_Reset().
- Perform NAND flash erase block operation using the function HAL_NAND_Erase_Block(). The erase block address information is contained in the Nand_Address_Typedef structure passed as parameter.
- Read the NAND flash status operation using the function HAL_NAND_Read_Status().
- You can also control the NAND device by calling the control APIs HAL_NAND_ECC_Enable()/ HAL_NAND_ECC_Disable() to respectively enable/disable the ECC code correction feature or the function HAL_NAND_GetECC() to get the ECC correction code.
- You can monitor the NAND device HAL state by calling the function HAL_NAND_GetState()

This driver is a set of generic APIs which handle standard NAND flash operations. If a NAND flash device contains different operations and/or implementations, it should be implemented separately.

41.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

This section contains the following APIs:

- ***HAL_NAND_Init()***
- ***HAL_NAND_DeInit()***
- ***HAL_NAND_MspInit()***
- ***HAL_NAND_MspDeInit()***
- ***HAL_NAND_IRQHandler()***
- ***HAL_NAND_ITCallback()***

41.2.3 NAND Input and Output functions

This section provides functions allowing to use and control the NAND memory

This section contains the following APIs:

- ***HAL_NAND_Read_ID()***
- ***HAL_NAND_Reset()***
- ***HAL_NAND_Read_Page()***
- ***HAL_NAND_Write_Page()***
- ***HAL_NAND_Read_SpareArea()***
- ***HAL_NAND_Write_SpareArea()***
- ***HAL_NAND_Erase_Block()***
- ***HAL_NAND_Read_Status()***
- ***HAL_NAND_Address_Inc()***

41.2.4 NAND Control functions

This subsection provides a set of functions allowing to control dynamically the NAND interface.

This section contains the following APIs:

- ***HAL_NAND_ECC_Enable()***
- ***HAL_NAND_ECC_Disable()***
- ***HAL_NAND_GetECC()***

41.2.5 NAND State functions

This subsection permits to get in run-time the status of the NAND controller and the data flow.

This section contains the following APIs:

- ***HAL_NAND_GetState()***
- ***HAL_NAND_Read_Status()***

41.2.6 Detailed description of functions

HAL_NAND_Init

Function name	**HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)**
Function description	Perform NAND memory Initialization sequence.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **ComSpace_Timing:** pointer to Common space timing structure • **AttSpace_Timing:** pointer to Attribute space timing structure
Return values	• **HAL:** status

HAL_NAND_DeInit

Function name	**HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)**
Function description	Perform NAND memory De-Initialization sequence.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• **HAL:** status

HAL_NAND_MspInit

Function name	**void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)**
Function description	Initialize the NAND MSP.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• **None:**

HAL_NAND_MspDeInit

Function name	**void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnand)**
Function description	DeInitialize the NAND MSP.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• **None:**

HAL_NAND_IRQHandler

Function name	**void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnand)**
Function description	This function handles NAND device interrupt request.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that

contains the configuration information for NAND module.

Return values
- **HAL:** status

HAL_NAND_ITCallback

Function name	**void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnand)**
Function description	NAND interrupt feature callback.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• **None:**

HAL_NAND_Read_ID

Function name	**HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)**
Function description	Read the NAND memory electronic signature.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **pNAND_ID:** NAND ID structure
Return values	• **HAL:** status

HAL_NAND_Reset

Function name	**HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnand)**
Function description	NAND memory reset.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• **HAL:** status

HAL_NAND_Read_Page

Function name	**HAL_StatusTypeDef HAL_NAND_Read_Page (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)**
Function description	Read Page(s) from NAND memory block.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **pAddress:** pointer to NAND address structure • **pBuffer:** pointer to destination read buffer • **NumPageToRead:** number of pages to read from block
Return values	• **HAL:** status

HAL_NAND_Write_Page

Function name	**HAL_StatusTypeDef HAL_NAND_Write_Page (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)**
Function description	Write Page(s) to NAND memory block.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **pAddress:** pointer to NAND address structure • **pBuffer:** pointer to source buffer to write • **NumPageToWrite:** number of pages to write to block
Return values	• **HAL:** status

HAL_NAND_Read_SpareArea

Function name	**HAL_StatusTypeDef HAL_NAND_Read_SpareArea (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)**
Function description	Read Spare area(s) from NAND memory.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **pAddress:** pointer to NAND address structure • **pBuffer:** pointer to source buffer to write • **NumSpareAreaToRead:** Number of spare area to read
Return values	• **HAL:** status

HAL_NAND_Write_SpareArea

Function name	**HAL_StatusTypeDef HAL_NAND_Write_SpareArea (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)**
Function description	Write Spare area(s) to NAND memory.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **pAddress:** pointer to NAND address structure • **pBuffer:** pointer to source buffer to write • **NumSpareAreaTowrite:** number of spare areas to write to block
Return values	• **HAL:** status

HAL_NAND_Erase_Block

Function name	**HAL_StatusTypeDef HAL_NAND_Erase_Block (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)**
Function description	NAND memory Block erase.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

- **pAddress:** pointer to NAND address structure

Return values
- **HAL:** status

HAL_NAND_Read_Status

Function name: **uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hnand)**

Function description: NAND memory read status.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values
- **NAND:** status

HAL_NAND_Address_Inc

Function name: **uint32_t HAL_NAND_Address_Inc (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)**

Function description: Increment the NAND memory address.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values
- **The:** new status of the increment address operation. It can be:
  - NAND_VALID_ADDRESS: When the new address is valid address
  - NAND_INVALID_ADDRESS: When the new address is invalid address

HAL_NAND_ECC_Enable

Function name: **HAL_StatusTypeDef HAL_NAND_ECC_Enable (NAND_HandleTypeDef * hnand)**

Function description: Enable dynamically NAND ECC feature.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values
- **HAL:** status

HAL_NAND_ECC_Disable

Function name: **HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)**

Function description: Disable dynamically NAND ECC feature.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values
- **HAL:** status

HAL_NAND_GetECC

Function name	**HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hnand, uint32_t * ECCval, uint32_t Timeout)**
Function description	Disable dynamically NAND ECC feature.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **ECCval:** pointer to ECC value • **Timeout:** maximum timeout to wait
Return values	• **HAL:** status

HAL_NAND_GetState

Function name	**HAL_NAND_StateTypeDef HAL_NAND_GetState (NAND_HandleTypeDef * hnand)**
Function description	Return the NAND handle state.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• **HAL:** state

HAL_NAND_Read_Status

Function name	**uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hnand)**
Function description	NAND memory read status.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• **NAND:** status

41.3 NAND Firmware driver defines

41.3.1 NAND

NAND Exported Macros

__HAL_NAND_RESET_HANDLE_STATE	**Description:** • Reset NAND handle state. **Parameters:** • __HANDLE__: specifies the NAND handle. **Return value:** • None

42 HAL NOR Generic Driver

42.1 NOR Firmware driver registers structures

42.1.1 NOR_IDTypeDef

Data Fields

- ***uint16_t Manufacturer_Code***
- ***uint16_t Device_Code1***
- ***uint16_t Device_Code2***
- ***uint16_t Device_Code3***

Field Documentation

- ***uint16_t NOR_IDTypeDef::Manufacturer_Code***
  Defines the device's manufacturer code used to identify the memory
- ***uint16_t NOR_IDTypeDef::Device_Code1***
- ***uint16_t NOR_IDTypeDef::Device_Code2***
- ***uint16_t NOR_IDTypeDef::Device_Code3***
  Defines the device's codes used to identify the memory. These codes can be accessed by performing read operations with specific control signals and addresses set.They can also be accessed by issuing an Auto Select command.

42.1.2 NOR_CFITypeDef

Data Fields

- ***uint16_t CFI_1***
- ***uint16_t CFI_2***
- ***uint16_t CFI_3***
- ***uint16_t CFI_4***

Field Documentation

- ***uint16_t NOR_CFITypeDef::CFI_1***
- ***uint16_t NOR_CFITypeDef::CFI_2***
- ***uint16_t NOR_CFITypeDef::CFI_3***
- ***uint16_t NOR_CFITypeDef::CFI_4***
  Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory.

42.1.3 NOR_HandleTypeDef

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NOR_StateTypeDef State***

Field Documentation

- ***FMC_NORSRAM_TypeDef* NOR_HandleTypeDef::Instance***
  Register base address

- ***FMC_NORSRAM_EXTENDED_TypeDef* NOR_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef NOR_HandleTypeDef::Init***
  NOR device control configuration parameters
- ***HAL_LockTypeDef NOR_HandleTypeDef::Lock***
  NOR locking object
- ***__IO HAL_NOR_StateTypeDef NOR_HandleTypeDef::State***
  NOR device access state

42.2 NOR Firmware driver API description

42.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function HAL_NOR_Init() with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function HAL_NOR_Read_ID(). The read information is stored in the NOR_ID_TypeDef structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions HAL_NOR_Read(), HAL_NOR_Program().
- Perform NOR flash erase block/chip operations using the functions HAL_NOR_Erase_Block() and HAL_NOR_Erase_Chip().
- Read the NOR flash CFI (common flash interface) IDs using the function HAL_NOR_Read_CFI(). The read information is stored in the NOR_CFI_TypeDef structure declared by the function caller.
- You can also control the NOR device by calling the control APIs HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable() to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function HAL_NOR_GetState()

This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- NOR_WRITE: NOR memory write data to specified address

42.2.2 NOR Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

This section contains the following APIs:

- ***HAL_NOR_Init()***
- ***HAL_NOR_DeInit()***
- ***HAL_NOR_MspInit()***
- ***HAL_NOR_MspDeInit()***
- ***HAL_NOR_MspWait()***

42.2.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

This section contains the following APIs:

- *HAL_NOR_Read_ID()*
- *HAL_NOR_ReturnToReadMode()*
- *HAL_NOR_Read()*
- *HAL_NOR_Program()*
- *HAL_NOR_ReadBuffer()*
- *HAL_NOR_ProgramBuffer()*
- *HAL_NOR_Erase_Block()*
- *HAL_NOR_Erase_Chip()*
- *HAL_NOR_Read_CFI()*

42.2.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

This section contains the following APIs:

- *HAL_NOR_WriteOperation_Enable()*
- *HAL_NOR_WriteOperation_Disable()*

42.2.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

This section contains the following APIs:

- *HAL_NOR_GetState()*
- *HAL_NOR_GetStatus()*

42.2.6 Detailed description of functions

HAL_NOR_Init

Function name	**HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)**
Function description	Perform the NOR memory Initialization sequence.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **Timing:** pointer to NOR control timing structure • **ExtTiming:** pointer to NOR extended mode timing structure
Return values	• **HAL:** status

HAL_NOR_DeInit

Function name	**HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)**
Function description	Perform NOR memory De-Initialization sequence.

Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	• **HAL:** status

HAL_NOR_MspInit

Function name	**void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)**
Function description	Initialize the NOR MSP.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	• **None:**

HAL_NOR_MspDeInit

Function name	**void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)**
Function description	DeInitialize the NOR MSP.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	• **None:**

HAL_NOR_MspWait

Function name	**void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)**
Function description	NOR MSP Wait for Ready/Busy signal.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **Timeout:** Maximum timeout value
Return values	• **None:**

HAL_NOR_Read_ID

Function name	**HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)**
Function description	Read NOR flash IDs.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **pNOR_ID:** : pointer to NOR ID structure
Return values	• **HAL:** status

HAL_NOR_ReturnToReadMode

Function name	**HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (NOR_HandleTypeDef * hnor)**
Function description	Return the NOR memory to Read mode.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that

contains the configuration information for NOR module.

Return values	• **HAL:** status

HAL_NOR_Read

Function name	**HAL_StatusTypeDef HAL_NOR_Read (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)**
Function description	Read data from NOR memory.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **pAddress:** pointer to Device address • **pData:** : pointer to read data
Return values	• **HAL:** status

HAL_NOR_Program

Function name	**HAL_StatusTypeDef HAL_NOR_Program (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)**
Function description	Program data to NOR memory.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **pAddress:** Device address • **pData:** : pointer to the data to write
Return values	• **HAL:** status

HAL_NOR_ReadBuffer

Function name	**HAL_StatusTypeDef HAL_NOR_ReadBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)**
Function description	Read a block of data from the FMC NOR memory.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **uwAddress:** NOR memory internal address to read from. • **pData:** pointer to the buffer that receives the data read from the NOR memory. • **uwBufferSize:** : number of Half word to read.
Return values	• **HAL:** status

HAL_NOR_ProgramBuffer

Function name	**HAL_StatusTypeDef HAL_NOR_ProgramBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)**
Function description	Write a half-word buffer to the FMC NOR memory.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

	• **uwAddress:** NOR memory internal address from which the data • **pData:** pointer to source data buffer. • **uwBufferSize:** number of Half words to write.
Return values	• **HAL:** status
Notes	• Some NOR memory need Address aligned to xx bytes (can be aligned to 64 bytes boundary for example). • The maximum buffer size allowed is NOR memory dependent (can be 64 Bytes max for example).

HAL_NOR_Erase_Block

Function name	**HAL_StatusTypeDef HAL_NOR_Erase_Block (NOR_HandleTypeDef * hnor, uint32_t BlockAddress, uint32_t Address)**
Function description	Erase the specified block of the NOR memory.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **BlockAddress:** : Block to erase address • **Address:** Device address
Return values	• **HAL:** status

HAL_NOR_Erase_Chip

Function name	**HAL_StatusTypeDef HAL_NOR_Erase_Chip (NOR_HandleTypeDef * hnor, uint32_t Address)**
Function description	Erase the entire NOR chip.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **Address:** : Device address
Return values	• **HAL:** status

HAL_NOR_Read_CFI

Function name	**HAL_StatusTypeDef HAL_NOR_Read_CFI (NOR_HandleTypeDef * hnor, NOR_CFITypeDef * pNOR_CFI)**
Function description	Read NOR flash CFI IDs.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **pNOR_CFI:** : pointer to NOR CFI IDs structure
Return values	• **HAL:** status

HAL_NOR_WriteOperation_Enable

Function name	**HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)**
Function description	Enable dynamically NOR write operation.

Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	• **HAL:** status

HAL_NOR_WriteOperation_Disable

Function name	**HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)**
Function description	Disable dynamically NOR write operation.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	• **HAL:** status

HAL_NOR_GetState

Function name	**HAL_NOR_StateTypeDef HAL_NOR_GetState (NOR_HandleTypeDef * hnor)**
Function description	Return the NOR controller handle state.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	• **NOR:** controller state

HAL_NOR_GetStatus

Function name	**HAL_NOR_StatusTypeDef HAL_NOR_GetStatus (NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)**
Function description	Return the NOR operation status.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **Address:** Device address • **Timeout:** NOR programming Timeout
Return values	• **NOR_Status:** The returned value can be: HAL_NOR_STATUS_SUCCESS, HAL_NOR_STATUS_ERROR or HAL_NOR_STATUS_TIMEOUT

42.3 NOR Firmware driver defines

42.3.1 NOR

NOR Exported Macros

__HAL_NOR_RESET_HANDLE_STATE

Description:

- Reset NOR handle state.

Parameters:

- __HANDLE__: NOR handle

Return value:

- None

43 HAL OPAMP Generic Driver

43.1 OPAMP Firmware driver registers structures

43.1.1 OPAMP_InitTypeDef

Data Fields

- ***uint32_t PowerSupplyRange***
- ***uint32_t PowerMode***
- ***uint32_t Mode***
- ***uint32_t InvertingInput***
- ***uint32_t NonInvertingInput***
- ***uint32_t PgaGain***
- ***uint32_t UserTrimming***
- ***uint32_t TrimmingValueP***
- ***uint32_t TrimmingValueN***
- ***uint32_t TrimmingValuePLowPower***
- ***uint32_t TrimmingValueNLowPower***

Field Documentation

- ***uint32_t OPAMP_InitTypeDef::PowerSupplyRange***
  Specifies the power supply range: above or under 2.4V. This parameter must be a value of ***OPAMP_PowerSupplyRange*** Caution: This parameter is common to all OPAMP instances: a modification of this parameter for the selected OPAMP impacts the other OPAMP instances.
- ***uint32_t OPAMP_InitTypeDef::PowerMode***
  Specifies the power mode Normal or Low-Power. This parameter must be a value of ***OPAMP_PowerMode***
- ***uint32_t OPAMP_InitTypeDef::Mode***
  Specifies the OPAMP mode This parameter must be a value of ***OPAMP_Mode*** mode is either Standalone, - Follower or PGA
- ***uint32_t OPAMP_InitTypeDef::InvertingInput***
  Specifies the inverting input in Standalone & PGA modesIn Standalone mode: i.e. when mode is OPAMP_STANDALONE_MODE & PGA mode: i.e. when mode is OPAMP_PGA_MODE This parameter must be a value of ***OPAMP_InvertingInput***In Follower mode i.e. when mode is OPAMP_FOLLOWER_MODE This parameter is Not Applicable
- ***uint32_t OPAMP_InitTypeDef::NonInvertingInput***
  Specifies the non inverting input of the opamp: This parameter must be a value of ***OPAMP_NonInvertingInput***
- ***uint32_t OPAMP_InitTypeDef::PgaGain***
  Specifies the gain in PGA mode i.e. when mode is OPAMP_PGA_MODE. This parameter must be a value of ***OPAMP_PgaGain*** (2, 4, 8 or 16)
- ***uint32_t OPAMP_InitTypeDef::UserTrimming***
  Specifies the trimming mode This parameter must be a value of ***OPAMP_UserTrimming*** UserTrimming is either factory or user trimming.
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueP***
  Specifies the offset trimming value (PMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31 16 is typical default value

- ***uint32_t OPAMP_InitTypeDef::TrimmingValueN***
  Specifies the offset trimming value (NMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValuePLowPower***
  Specifies the offset trimming value (PMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueNLowPower***
  Specifies the offset trimming value (NMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31 16 is typical default value

43.1.2 OPAMP_HandleTypeDef

Data Fields

- ***OPAMP_TypeDef * Instance***
- ***OPAMP_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_OPAMP_StateTypeDef State***

Field Documentation

- ***OPAMP_TypeDef* OPAMP_HandleTypeDef::Instance***
  OPAMP instance's registers base address
- ***OPAMP_InitTypeDef OPAMP_HandleTypeDef::Init***
  OPAMP required parameters
- ***HAL_StatusTypeDef OPAMP_HandleTypeDef::Status***
  OPAMP peripheral status
- ***HAL_LockTypeDef OPAMP_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_OPAMP_StateTypeDef OPAMP_HandleTypeDef::State***
  OPAMP communication state

43.2 OPAMP Firmware driver API description

43.2.1 OPAMP Peripheral Features

The device integrates 1 or 2 operational amplifiers OPAMP1 & OPAMP2

1. The OPAMP(s) provide(s) several exclusive running modes.
   - 1 OPAMP: STM32L431xx STM32L432xx STM32L433xx STM32L442xx STM32L443xx
   - 2 OPAMP: STM32L471xx STM32L475xx STM32L476xx STM32L485xx STM32L486xx
2. The OPAMP(s) provide(s) several exclusive running modes.
   - Standalone mode
   - Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
   - Follower mode
3. All OPAMP (same for all OPAMPs) can operate in
   - Either Low range (VDDA < 2.4V) power supply
   - Or High range (VDDA > 2.4V) power supply
4. Each OPAMP(s) can be configured in normal and low power mode.
5. The OPAMP(s) provide(s) calibration capabilities.
   - Calibration aims at correcting some offset for running mode.

- The OPAMP uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
- The user defined settings can be figured out using self calibration handled by HAL_OPAMP_SelfCalibrate, HAL_OPAMPEx_SelfCalibrateAll
- HAL_OPAMP_SelfCalibrate:
  - Runs automatically the calibration.
  - Enables the user trimming mode
  - Updates the init structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)
  - HAL_OPAMPEx_SelfCalibrateAll runs calibration of all OPAMPs in parallel to save search time.

6. Running mode: Standalone mode
   - Gain is set externally (gain depends on external loads).
   - Follower mode also possible externally by connecting the inverting input to the output.
7. Running mode: Follower mode
   - No Inverting Input is connected.
8. Running mode: Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
   - The OPAMP(s) output(s) can be internally connected to resistor feedback output.
   - OPAMP gain is either 2, 4, 8 or 16.
9. The OPAMPs inverting input can be selected according to the Reference Manual "OPAMP function description" chapter.
10. The OPAMPs non inverting input can be selected according to the Reference Manual "OPAMP function description" chapter.

43.2.2 How to use this driver

Power supply range

To run in low power mode:

1. Configure the OPAMP using HAL_OPAMP_Init() function:
   - Select OPAMP_POWERSUPPLY_LOW (VDDA lower than 2.4V)
   - Otherwise select OPAMP_POWERSUPPLY_HIGH (VDDA higher than 2.4V)

Low / normal power mode

To run in low power mode:

1. Configure the OPAMP using HAL_OPAMP_Init() function:
   - Select OPAMP_POWERMODE_LOWPOWER
   - Otherwise select OPAMP_POWERMODE_NORMAL

Calibration

To run the OPAMP calibration self calibration:

1. Start calibration using HAL_OPAMP_SelfCalibrate. Store the calibration results.

Running mode

To use the OPAMP, perform the following steps:

1. Fill in the HAL_OPAMP_MspInit() to

   - Enable the OPAMP Peripheral clock using macro __HAL_RCC_OPAMP_CLK_ENABLE()
   - Configure the OPAMP input AND output in analog mode using HAL_GPIO_Init() to map the OPAMP output to the GPIO pin.
2. Configure the OPAMP using HAL_OPAMP_Init() function:
   - Select the mode
   - Select the inverting input
   - Select the non-inverting input
   - If PGA mode is enabled, Select if inverting input is connected.
   - Select either factory or user defined trimming mode.
   - If the user-defined trimming mode is enabled, select PMOS & NMOS trimming values (typically values set by HAL_OPAMP_SelfCalibrate function).
3. Enable the OPAMP using HAL_OPAMP_Start() function.
4. Disable the OPAMP using HAL_OPAMP_Stop() function.
5. Lock the OPAMP in running mode using HAL_OPAMP_Lock() function. Caution: On STM32L4, HAL OPAMP lock is software lock only (not hardware lock as on some other STM32 devices)
6. If needed, unlock the OPAMP using HAL_OPAMPEx_Unlock() function.

Running mode: change of configuration while OPAMP ON

To Re-configure OPAMP when OPAMP is ON (change on the fly)

1. If needed, fill in the HAL_OPAMP_MspInit()
   - This is the case for instance if you wish to use new OPAMP I/O
2. Configure the OPAMP using HAL_OPAMP_Init() function:
   - As in configure case, select first the parameters you wish to modify.
3. Change from low power mode to normal power mode (& vice versa) requires first HAL_OPAMP_DeInit() (force OPAMP OFF) and then HAL_OPAMP_Init(). In other words, of OPAMP is ON, HAL_OPAMP_Init can NOT change power mode alone.

43.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_OPAMP_Init()***
- ***HAL_OPAMP_DeInit()***
- ***HAL_OPAMP_MspInit()***
- ***HAL_OPAMP_MspDeInit()***

43.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the OPAMP start, stop and calibration actions.

This section contains the following APIs:

- ***HAL_OPAMP_Start()***
- ***HAL_OPAMP_Stop()***
- ***HAL_OPAMP_SelfCalibrate()***

43.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the OPAMP data transfers.

This section contains the following APIs:

- ***HAL_OPAMP_Lock()***
- ***HAL_OPAMP_GetTrimOffset()***

43.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_OPAMP_GetState()*

43.2.7 Detailed description of functions

HAL_OPAMP_Init

Function name	**HAL_StatusTypeDef HAL_OPAMP_Init (OPAMP_HandleTypeDef * hopamp)**
Function description	Initializes the OPAMP according to the specified parameters in the OPAMP_InitTypeDef and initialize the associated handle.
Parameters	• **hopamp:** OPAMP handle
Return values	• **HAL:** status
Notes	• If the selected opamp is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_OPAMP_DeInit

Function name	**HAL_StatusTypeDef HAL_OPAMP_DeInit (OPAMP_HandleTypeDef * hopamp)**
Function description	DeInitialize the OPAMP peripheral.
Parameters	• **hopamp:** OPAMP handle
Return values	• **HAL:** status
Notes	• Deinitialization can be performed if the OPAMP configuration is locked. (the lock is SW in L4)

HAL_OPAMP_MspInit

Function name	**void HAL_OPAMP_MspInit (OPAMP_HandleTypeDef * hopamp)**
Function description	Initialize the OPAMP MSP.
Parameters	• **hopamp:** OPAMP handle
Return values	• **None:**

HAL_OPAMP_MspDeInit

Function name	**void HAL_OPAMP_MspDeInit (OPAMP_HandleTypeDef * hopamp)**
Function description	DeInitialize OPAMP MSP.
Parameters	• **hopamp:** OPAMP handle
Return values	• **None:**

HAL_OPAMP_Start

Function name	**HAL_StatusTypeDef HAL_OPAMP_Start (OPAMP_HandleTypeDef * hopamp)**
Function description	Start the OPAMP.
Parameters	• **hopamp:** OPAMP handle
Return values	• **HAL:** status

HAL_OPAMP_Stop

Function name	**HAL_StatusTypeDef HAL_OPAMP_Stop (OPAMP_HandleTypeDef * hopamp)**
Function description	Stop the OPAMP.
Parameters	• **hopamp:** OPAMP handle
Return values	• **HAL:** status

HAL_OPAMP_SelfCalibrate

Function name	**HAL_StatusTypeDef HAL_OPAMP_SelfCalibrate (OPAMP_HandleTypeDef * hopamp)**
Function description	Run the self calibration of one OPAMP.
Parameters	• **hopamp:** handle
Return values	• **Updated:** offset trimming values (PMOS & NMOS), user trimming is enabled • **HAL:** status
Notes	• Calibration is performed in the mode specified in OPAMP init structure (mode normal or low-power). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated. • Calibration runs about 10 ms.

HAL_OPAMP_Lock

Function name	**HAL_StatusTypeDef HAL_OPAMP_Lock (OPAMP_HandleTypeDef * hopamp)**
Function description	Lock the selected OPAMP configuration.
Parameters	• **hopamp:** OPAMP handle
Return values	• **HAL:** status
Notes	• On STM32L4, HAL OPAMP lock is software lock only (in contrast of hardware lock available on some other STM32 devices).

HAL_OPAMP_GetTrimOffset

Function name	**HAL_OPAMP_TrimmingValueTypeDef HAL_OPAMP_GetTrimOffset (OPAMP_HandleTypeDef * hopamp, uint32_t trimmingoffset)**

Function description	Return the OPAMP factory trimming value.
Parameters	• **hopamp:** : OPAMP handle • **trimmingoffset:** : Trimming offset (P or N) This parameter must be a value of OPAMP Factory Trimming
Return values	• **Trimming:** value (P or N): range: 0->31 or OPAMP_FACTORYTRIMMING_DUMMY if trimming value is not available
Notes	• On STM32L4 OPAMP, user can retrieve factory trimming if OPAMP has never been set to user trimming before. Therefore, this function must be called when OPAMP init parameter "UserTrimming" is set to trimming factory, and before OPAMP calibration (function "HAL_OPAMP_SelfCalibrate()"). Otherwise, factory trimming value cannot be retrieved and error status is returned. • Calibration parameter retrieved is corresponding to the mode specified in OPAMP init structure (mode normal or low-power). To retrieve calibration parameters for both modes, repeat this function after OPAMP init structure accordingly updated.

HAL_OPAMP_GetState

Function name	**HAL_OPAMP_StateTypeDef HAL_OPAMP_GetState (OPAMP_HandleTypeDef * hopamp)**
Function description	Return the OPAMP handle state.
Parameters	• **hopamp:** : OPAMP handle
Return values	• **HAL:** state

43.3 OPAMP Firmware driver defines

43.3.1 OPAMP

OPAMP Exported Macros

__HAL_OPAMP_RESET_HANDLE_STATE	**Description:** • Reset OPAMP handle state. **Parameters:** • __HANDLE__: OPAMP handle. **Return value:** • None

OPAMP Factory Trimming

OPAMP_FACTORYTRIMMING_DUMMY	Dummy value if trimming value could not be retrieved
OPAMP_FACTORYTRIMMING_N	Offset trimming N
OPAMP_FACTORYTRIMMING_P	Offset trimming P

OPAMP Inverting Input

OPAMP_INVERTINGINPUT_IO0	OPAMP inverting input connected to dedicated IO pin low-leakage
OPAMP_INVERTINGINPUT_IO1	OPAMP inverting input connected to alternative IO pin available on some device packages
OPAMP_INVERTINGINPUT_CONNECT_NO	OPAMP inverting input not connected externally (PGA mode only)

OPAMP Mode

OPAMP_STANDALONE_MODE	standalone mode
OPAMP_PGA_MODE	PGA mode
OPAMP_FOLLOWER_MODE	follower mode

OPAMP Non Inverting Input

OPAMP_NONINVERTINGINPUT_IO0	OPAMP non-inverting input connected to dedicated IO pin
OPAMP_NONINVERTINGINPUT_DAC_CH	OPAMP non-inverting input connected internally to DAC channel

OPAMP Pga Gain

OPAMP_PGA_GAIN_2	PGA gain = 2
OPAMP_PGA_GAIN_4	PGA gain = 4
OPAMP_PGA_GAIN_8	PGA gain = 8
OPAMP_PGA_GAIN_16	PGA gain = 16

OPAMP PowerMode

OPAMP_POWERMODE_NORMAL

OPAMP_POWERMODE_LOWPOWER

OPAMP PowerSupplyRange

OPAMP_POWERSUPPLY_LOW	Power supply range low (VDDA lower than 2.4V)
OPAMP_POWERSUPPLY_HIGH	Power supply range high (VDDA higher than 2.4V)

OPAMP User Trimming

OPAMP_TRIMMING_FACTORY	Factory trimming
OPAMP_TRIMMING_USER	User trimming

44 HAL OPAMP Extension Driver

44.1 OPAMPEx Firmware driver API description

44.1.1 Extended IO operation functions

- OPAMP Self calibration.

44.1.2 Peripheral Control functions

- OPAMP unlock.

This section contains the following APIs:

- *HAL_OPAMPEx_Unlock()*

44.1.3 Detailed description of functions

HAL_OPAMPEx_SelfCalibrateAll

Function name	**HAL_StatusTypeDef HAL_OPAMPEx_SelfCalibrateAll (OPAMP_HandleTypeDef * hopamp1, OPAMP_HandleTypeDef * hopamp2)**
Function description	Run the self calibration of the 2 OPAMPs in parallel.
Parameters	• **hopamp1:** handle • **hopamp2:** handle
Return values	• **HAL:** status
Notes	• Trimming values (PMOS & NMOS) are updated and user trimming is enabled is calibration is successful. • Calibration is performed in the mode specified in OPAMP init structure (mode normal or low-power). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated. • Calibration runs about 10 ms (5 dichotomy steps, repeated for P and N transistors: 10 steps with 1 ms for each step).

HAL_OPAMPEx_Unlock

Function name	**HAL_StatusTypeDef HAL_OPAMPEx_Unlock (OPAMP_HandleTypeDef * hopamp)**
Function description	Unlock the selected OPAMP configuration.
Parameters	• **hopamp:** OPAMP handle
Return values	• **HAL:** status
Notes	• This function must be called only when OPAMP is in state "locked".

45 HAL OSPI Generic Driver

45.1 OSPI Firmware driver registers structures

45.1.1 OSPI_InitTypeDef

Data Fields

- ***uint32_t FifoThreshold***
- ***uint32_t DualQuad***
- ***uint32_t MemoryType***
- ***uint32_t DeviceSize***
- ***uint32_t ChipSelectHighTime***
- ***uint32_t FreeRunningClock***
- ***uint32_t ClockMode***
- ***uint32_t WrapSize***
- ***uint32_t ClockPrescaler***
- ***uint32_t SampleShifting***
- ***uint32_t DelayHoldQuarterCycle***
- ***uint32_t ChipSelectBoundary***

Field Documentation

- ***uint32_t OSPI_InitTypeDef::FifoThreshold***
- ***uint32_t OSPI_InitTypeDef::DualQuad***
- ***uint32_t OSPI_InitTypeDef::MemoryType***
- ***uint32_t OSPI_InitTypeDef::DeviceSize***
- ***uint32_t OSPI_InitTypeDef::ChipSelectHighTime***
- ***uint32_t OSPI_InitTypeDef::FreeRunningClock***
- ***uint32_t OSPI_InitTypeDef::ClockMode***
- ***uint32_t OSPI_InitTypeDef::WrapSize***
- ***uint32_t OSPI_InitTypeDef::ClockPrescaler***
- ***uint32_t OSPI_InitTypeDef::SampleShifting***
- ***uint32_t OSPI_InitTypeDef::DelayHoldQuarterCycle***
- ***uint32_t OSPI_InitTypeDef::ChipSelectBoundary***

45.1.2 OSPI_HandleTypeDef

Data Fields

- ***OCTOSPI_TypeDef * Instance***
- ***OSPI_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***__IO uint32_t XferSize***
- ***__IO uint32_t XferCount***
- ***DMA_HandleTypeDef * hdma***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Timeout***

Field Documentation

- ***OCTOSPI_TypeDef* OSPI_HandleTypeDef::Instance***
- ***OSPI_InitTypeDef OSPI_HandleTypeDef::Init***

- ***uint8_t* OSPI_HandleTypeDef::pBuffPtr***
- ***__IO uint32_t OSPI_HandleTypeDef::XferSize***
- ***__IO uint32_t OSPI_HandleTypeDef::XferCount***
- ***DMA_HandleTypeDef* OSPI_HandleTypeDef::hdma***
- ***__IO uint32_t OSPI_HandleTypeDef::State***
- ***__IO uint32_t OSPI_HandleTypeDef::ErrorCode***
- ***uint32_t OSPI_HandleTypeDef::Timeout***

45.1.3 OSPI_RegularCmdTypeDef

Data Fields

- ***uint32_t OperationType***
- ***uint32_t FlashId***
- ***uint32_t Instruction***
- ***uint32_t InstructionMode***
- ***uint32_t InstructionSize***
- ***uint32_t InstructionDtrMode***
- ***uint32_t Address***
- ***uint32_t AddressMode***
- ***uint32_t AddressSize***
- ***uint32_t AddressDtrMode***
- ***uint32_t AlternateBytes***
- ***uint32_t AlternateBytesMode***
- ***uint32_t AlternateBytesSize***
- ***uint32_t AlternateBytesDtrMode***
- ***uint32_t DataMode***
- ***uint32_t NbData***
- ***uint32_t DataDtrMode***
- ***uint32_t DummyCycles***
- ***uint32_t DQSMode***
- ***uint32_t SIOOMode***

Field Documentation

- ***uint32_t OSPI_RegularCmdTypeDef::OperationType***
- ***uint32_t OSPI_RegularCmdTypeDef::FlashId***
- ***uint32_t OSPI_RegularCmdTypeDef::Instruction***
- ***uint32_t OSPI_RegularCmdTypeDef::InstructionMode***
- ***uint32_t OSPI_RegularCmdTypeDef::InstructionSize***
- ***uint32_t OSPI_RegularCmdTypeDef::InstructionDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::Address***
- ***uint32_t OSPI_RegularCmdTypeDef::AddressMode***
- ***uint32_t OSPI_RegularCmdTypeDef::AddressSize***
- ***uint32_t OSPI_RegularCmdTypeDef::AddressDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytes***
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytesMode***
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytesSize***
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytesDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::DataMode***
- ***uint32_t OSPI_RegularCmdTypeDef::NbData***
- ***uint32_t OSPI_RegularCmdTypeDef::DataDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::DummyCycles***
- ***uint32_t OSPI_RegularCmdTypeDef::DQSMode***

- ***uint32_t OSPI_RegularCmdTypeDef::SIOOMode***

45.1.4 OSPI_HyperbusCfgTypeDef

Data Fields

- ***uint32_t RWRecoveryTime***
- ***uint32_t AccessTime***
- ***uint32_t WriteZeroLatency***
- ***uint32_t LatencyMode***

Field Documentation

- ***uint32_t OSPI_HyperbusCfgTypeDef::RWRecoveryTime***
- ***uint32_t OSPI_HyperbusCfgTypeDef::AccessTime***
- ***uint32_t OSPI_HyperbusCfgTypeDef::WriteZeroLatency***
- ***uint32_t OSPI_HyperbusCfgTypeDef::LatencyMode***

45.1.5 OSPI_HyperbusCmdTypeDef

Data Fields

- ***uint32_t AddressSpace***
- ***uint32_t Address***
- ***uint32_t AddressSize***
- ***uint32_t NbData***
- ***uint32_t DQSMode***

Field Documentation

- ***uint32_t OSPI_HyperbusCmdTypeDef::AddressSpace***
- ***uint32_t OSPI_HyperbusCmdTypeDef::Address***
- ***uint32_t OSPI_HyperbusCmdTypeDef::AddressSize***
- ***uint32_t OSPI_HyperbusCmdTypeDef::NbData***
- ***uint32_t OSPI_HyperbusCmdTypeDef::DQSMode***

45.1.6 OSPI_AutoPollingTypeDef

Data Fields

- ***uint32_t Match***
- ***uint32_t Mask***
- ***uint32_t MatchMode***
- ***uint32_t AutomaticStop***
- ***uint32_t Interval***

Field Documentation

- ***uint32_t OSPI_AutoPollingTypeDef::Match***
- ***uint32_t OSPI_AutoPollingTypeDef::Mask***
- ***uint32_t OSPI_AutoPollingTypeDef::MatchMode***
- ***uint32_t OSPI_AutoPollingTypeDef::AutomaticStop***
- ***uint32_t OSPI_AutoPollingTypeDef::Interval***

45.1.7 OSPI_MemoryMappedTypeDef

Data Fields

- ***uint32_t TimeOutActivation***
- ***uint32_t TimeOutPeriod***

Field Documentation

- ***uint32_t OSPI_MemoryMappedTypeDef::TimeOutActivation***
- ***uint32_t OSPI_MemoryMappedTypeDef::TimeOutPeriod***

45.1.8 OSPIM_CfgTypeDef

Data Fields

- ***uint32_t ClkPort***
- ***uint32_t DQSPort***
- ***uint32_t NCSPort***
- ***uint32_t IOLowPort***
- ***uint32_t IOHighPort***

Field Documentation

- ***uint32_t OSPIM_CfgTypeDef::ClkPort***
- ***uint32_t OSPIM_CfgTypeDef::DQSPort***
- ***uint32_t OSPIM_CfgTypeDef::NCSPort***
- ***uint32_t OSPIM_CfgTypeDef::IOLowPort***
- ***uint32_t OSPIM_CfgTypeDef::IOHighPort***

45.2 OSPI Firmware driver API description

45.2.1 How to use this driver

Initialization

1. As prerequisite, fill in the HAL_OSPI_MspInit():
   - Enable OctoSPI and OctoSPIM clocks interface with __HAL_RCC_OSPIx_CLK_ENABLE().
   - Reset OctoSPI IP with __HAL_RCC_OSPIx_FORCE_RESET() and __HAL_RCC_OSPIx_RELEASE_RESET().
   - Enable the clocks for the OctoSPI GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
   - Configure these OctoSPI pins in alternate mode using HAL_GPIO_Init().
   - If interrupt or DMA mode is used, enable and configure OctoSPI global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
   - If DMA mode is used, enable the clocks for the OctoSPI DMA channel with __HAL_RCC_DMAx_CLK_ENABLE(), configure DMA with HAL_DMA_Init(), link it with OctoSPI handle using __HAL_LINKDMA(), enable and configure DMA channel global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Configure the fifo threshold, the dual-quad mode, the memory type, the device size, the CS high time, the free running clock, the clock mode, the wrap size, the clock prescaler, the sample shifting, the hold delay and the CS boundary using the HAL_OSPI_Init() function.
3. When using Hyperbus, configure the RW recovery time, the access time, the write latency and the latency mode unsing the HAL_OSPI_HyperbusCfg() function.

Indirect functional mode

1. In regular mode, configure the command sequence using the HAL_OSPI_Command() or HAL_OSPI_Command_IT() functions:

   - Instruction phase: the mode used and if present the size, the instruction opcode and the DTR mode.
   - Address phase: the mode used and if present the size, the address value and the DTR mode.
   - Alternate-bytes phase: the mode used and if present the size, the alternate bytes values and the DTR mode.
   - Dummy-cycles phase: the number of dummy cycles (mode used is same as data phase).
   - Data phase: the mode used and if present the number of bytes and the DTR mode.
   - Data strobe (DQS) mode: the activation (or not) of this mode
   - Sending Instruction Only Once (SIOO) mode: the activation (or not) of this mode.
   - Flash identifier: in dual-quad mode, indicates which flash is concerned
   - Operation type: always common configuration
2. In Hyperbus mode, configure the command sequence using the HAL_OSPI_HyperbusCmd() function:
   - Address space: indicate if the access will be done in register or memory
   - Address size
   - Number of data
   - Data strobe (DQS) mode: the activation (or not) of this mode
3. If no data is required for the command (only for regular mode, not for Hyperbus mode), it is sent directly to the memory:
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_OSPI_CmdCpltCallback() will be called when the transfer is complete.
4. For the indirect write mode, use HAL_OSPI_Transmit(), HAL_OSPI_Transmit_DMA() or HAL_OSPI_Transmit_IT() after the command configuration:
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_OSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_OSPI_TxCpltCallback() will be called when the transfer is complete.
   - In DMA mode, HAL_OSPI_TxHalfCpltCallback() will be called at the half transfer and HAL_OSPI_TxCpltCallback() will be called when the transfer is complete.
5. For the indirect read mode, use HAL_OSPI_Receive(), HAL_OSPI_Receive_DMA() or HAL_OSPI_Receive_IT() after the command configuration:
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_OSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_OSPI_RxCpltCallback() will be called when the transfer is complete.
   - In DMA mode, HAL_OSPI_RxHalfCpltCallback() will be called at the half transfer and HAL_OSPI_RxCpltCallback() will be called when the transfer is complete.

Auto-polling functional mode

1. Configure the command sequence by the same way than the indirect mode
2. Configure the auto-polling functional mode using the HAL_OSPI_AutoPolling() or HAL_OSPI_AutoPolling_IT() functions:
   - The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.
3. After the configuration:
   - In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
   - In interrupt mode, HAL_OSPI_StatusMatchCallback() will be called each time the status match is reached.

Memory-mapped functional mode

1. Configure the command sequence by the same way than the indirect mode except for the operation type in regular mode:
   - Operation type equals to read configuration: the command configuration applies to read access in memory-mapped mode
   - Operation type equals to write configuration: the command configuration applies to write access in memory-mapped mode
   - Both read and write configuration should be performed before activating memory-mapped mode
2. Configure the memory-mapped functional mode using the HAL_OSPI_MemoryMapped() functions:
   - The timeout activation and the timeout period.
3. After the configuration, the OctoSPI will be used as soon as an access on the AHB is done on the address range. HAL_OSPI_TimeOutCallback() will be called when the timeout expires.

Errors management and abort functionality

1. HAL_OSPI_GetError() function gives the error raised during the last operation.
2. HAL_OSPI_Abort() and HAL_OSPI_AbortIT() functions aborts any on-going operation and flushes the fifo:
   - In polling mode, the output of the function is done when the transfer complete bit is set and the busy bit cleared.
   - In interrupt mode, HAL_OSPI_AbortCpltCallback() will be called when the transfer complete bit is set.

Control functions

1. HAL_OSPI_GetState() function gives the current state of the HAL OctoSPI driver.
2. HAL_OSPI_SetTimeout() function configures the timeout value used in the driver.
3. HAL_OSPI_SetFifoThreshold() function configures the threshold on the Fifo of the OSPI IP.
4. HAL_OSPI_GetFifoThreshold() function gives the current of the Fifo's threshold

IO manager configuration functions

1. HAL_OSPIM_Config() function configures the IO manager for the OctoSPI instance.

45.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to:

- Initialize the OctoSPI.
- De-initialize the OctoSPI.

This section contains the following APIs:

- ***HAL_OSPI_Init()***
- ***HAL_OSPI_MspInit()***
- ***HAL_OSPI_DeInit()***
- ***HAL_OSPI_MspDeInit()***

45.2.3 IO operation functions

This subsection provides a set of functions allowing to:

- Handle the interrupts.
- Handle the command sequence (regular and Hyperbus).

- Handle the Hyperbus configuration.
- Transmit data in blocking, interrupt or DMA mode.
- Receive data in blocking, interrupt or DMA mode.
- Manage the auto-polling functional mode.
- Manage the memory-mapped functional mode.

This section contains the following APIs:

- ***HAL_OSPI_IRQHandler()***
- ***HAL_OSPI_Command()***
- ***HAL_OSPI_Command_IT()***
- ***HAL_OSPI_HyperbusCfg()***
- ***HAL_OSPI_HyperbusCmd()***
- ***HAL_OSPI_Transmit()***
- ***HAL_OSPI_Receive()***
- ***HAL_OSPI_Transmit_IT()***
- ***HAL_OSPI_Receive_IT()***
- ***HAL_OSPI_Transmit_DMA()***
- ***HAL_OSPI_Receive_DMA()***
- ***HAL_OSPI_AutoPolling()***
- ***HAL_OSPI_AutoPolling_IT()***
- ***HAL_OSPI_MemoryMapped()***
- ***HAL_OSPI_ErrorCallback()***
- ***HAL_OSPI_AbortCpltCallback()***
- ***HAL_OSPI_FifoThresholdCallback()***
- ***HAL_OSPI_CmdCpltCallback()***
- ***HAL_OSPI_RxCpltCallback()***
- ***HAL_OSPI_TxCpltCallback()***
- ***HAL_OSPI_RxHalfCpltCallback()***
- ***HAL_OSPI_TxHalfCpltCallback()***
- ***HAL_OSPI_StatusMatchCallback()***
- ***HAL_OSPI_TimeOutCallback()***

45.2.4 Peripheral Control and State functions

This subsection provides a set of functions allowing to:

- Check in run-time the state of the driver.
- Check the error code set during last operation.
- Abort any operation.
- Manage the Fifo threshold.
- Configure the timeout duration used in the driver.

This section contains the following APIs:

- ***HAL_OSPI_Abort()***
- ***HAL_OSPI_Abort_IT()***
- ***HAL_OSPI_SetFifoThreshold()***
- ***HAL_OSPI_GetFifoThreshold()***
- ***HAL_OSPI_SetTimeout()***
- ***HAL_OSPI_GetError()***
- ***HAL_OSPI_GetState()***

45.2.5 IO Manager configuration function

This subsection provides a set of functions allowing to:

- Configure the IO manager.

This section contains the following APIs:

- *HAL_OSPIM_Config()*

45.2.6 Detailed description of functions

HAL_OSPI_Init

Function name	**HAL_StatusTypeDef HAL_OSPI_Init (OSPI_HandleTypeDef * hospi)**
Function description	Initialize the OSPI mode according to the specified parameters in the OSPI_InitTypeDef and initialize the associated handle.
Parameters	• **hospi:** : OSPI handle
Return values	• **HAL:** status

HAL_OSPI_MspInit

Function name	**void HAL_OSPI_MspInit (OSPI_HandleTypeDef * hospi)**
Function description	Initialize the OSPI MSP.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_DeInit

Function name	**HAL_StatusTypeDef HAL_OSPI_DeInit (OSPI_HandleTypeDef * hospi)**
Function description	De-Initialize the OSPI peripheral.
Parameters	• **hospi:** : OSPI handle
Return values	• **HAL:** status

HAL_OSPI_MspDeInit

Function name	**void HAL_OSPI_MspDeInit (OSPI_HandleTypeDef * hospi)**
Function description	DeInitialize the OSPI MSP.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_IRQHandler

Function name	**void HAL_OSPI_IRQHandler (OSPI_HandleTypeDef * hospi)**
Function description	Handle OSPI interrupt request.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_Command

Function name	**HAL_StatusTypeDef HAL_OSPI_Command (OSPI_HandleTypeDef * hospi, OSPI_RegularCmdTypeDef * cmd, uint32_t Timeout)**
Function description	Set the command configuration.
Parameters	• **hospi:** : OSPI handle • **cmd:** : structure that contains the command configuration information • **Timeout:** : Timeout duration
Return values	• **HAL:** status

HAL_OSPI_Command_IT

Function name	**HAL_StatusTypeDef HAL_OSPI_Command_IT (OSPI_HandleTypeDef * hospi, OSPI_RegularCmdTypeDef * cmd)**
Function description	Set the command configuration in interrupt mode.
Parameters	• **hospi:** : OSPI handle • **cmd:** : structure that contains the command configuration information
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Read or Write Modes

HAL_OSPI_HyperbusCfg

Function name	**HAL_StatusTypeDef HAL_OSPI_HyperbusCfg (OSPI_HandleTypeDef * hospi, OSPI_HyperbusCfgTypeDef * cfg, uint32_t Timeout)**
Function description	Configure the Hyperbus parameters.
Parameters	• **hospi:** : OSPI handle • **cfg:** : Structure containing the Hyperbus configuration • **Timeout:** : Timeout duration
Return values	• **HAL:** status

HAL_OSPI_HyperbusCmd

Function name	**HAL_StatusTypeDef HAL_OSPI_HyperbusCmd (OSPI_HandleTypeDef * hospi, OSPI_HyperbusCmdTypeDef * cmd, uint32_t Timeout)**
Function description	Set the Hyperbus command configuration.
Parameters	• **hospi:** : OSPI handle • **cmd:** : Structure containing the Hyperbus command • **Timeout:** : Timeout duration
Return values	• **HAL:** status

HAL_OSPI_Transmit

Function name	**HAL_StatusTypeDef HAL_OSPI_Transmit (OSPI_HandleTypeDef * hospi, uint8_t * pData, uint32_t Timeout)**
Function description	Transmit an amount of data in blocking mode.
Parameters	• **hospi:** : OSPI handle • **pData:** : pointer to data buffer • **Timeout:** : Timeout duration
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Write Mode

HAL_OSPI_Receive

Function name	**HAL_StatusTypeDef HAL_OSPI_Receive (OSPI_HandleTypeDef * hospi, uint8_t * pData, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **hospi:** : OSPI handle • **pData:** : pointer to data buffer • **Timeout:** : Timeout duration
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Read Mode

HAL_OSPI_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_OSPI_Transmit_IT (OSPI_HandleTypeDef * hospi, uint8_t * pData)**
Function description	Send an amount of data in non-blocking mode with interrupt.
Parameters	• **hospi:** : OSPI handle • **pData:** : pointer to data buffer
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Write Mode

HAL_OSPI_Receive_IT

Function name	**HAL_StatusTypeDef HAL_OSPI_Receive_IT (OSPI_HandleTypeDef * hospi, uint8_t * pData)**
Function description	Receive an amount of data in non-blocking mode with interrupt.
Parameters	• **hospi:** : OSPI handle • **pData:** : pointer to data buffer
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Read Mode

HAL_OSPI_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_OSPI_Transmit_DMA (OSPI_HandleTypeDef * hospi, uint8_t * pData)**
Function description	Send an amount of data in non-blocking mode with DMA.
Parameters	• **hospi:** : OSPI handle • **pData:** : pointer to data buffer
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Write Mode • If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword • If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_OSPI_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_OSPI_Receive_DMA (OSPI_HandleTypeDef * hospi, uint8_t * pData)**
Function description	Receive an amount of data in non-blocking mode with DMA.
Parameters	• **hospi:** : OSPI handle • **pData:** : pointer to data buffer.
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Read Mode • If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword • If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_OSPI_AutoPolling

Function name	**HAL_StatusTypeDef HAL_OSPI_AutoPolling (OSPI_HandleTypeDef * hospi, OSPI_AutoPollingTypeDef * cfg, uint32_t Timeout)**
Function description	Configure the OSPI Automatic Polling Mode in blocking mode.
Parameters	• **hospi:** : OSPI handle • **cfg:** : structure that contains the polling configuration information. • **Timeout:** : Timeout duration
Return values	• **HAL:** status
Notes	• This function is used only in Automatic Polling Mode

HAL_OSPI_AutoPolling_IT

Function name	**HAL_StatusTypeDef HAL_OSPI_AutoPolling_IT (OSPI_HandleTypeDef * hospi, OSPI_AutoPollingTypeDef * cfg)**

Function description	Configure the OSPI Automatic Polling Mode in non-blocking mode.
Parameters	• **hospi:** : OSPI handle • **cfg:** : structure that contains the polling configuration information.
Return values	• **HAL:** status
Notes	• This function is used only in Automatic Polling Mode

HAL_OSPI_MemoryMapped

Function name	**HAL_StatusTypeDef HAL_OSPI_MemoryMapped (OSPI_HandleTypeDef * hospi, OSPI_MemoryMappedTypeDef * cfg)**
Function description	Configure the Memory Mapped mode.
Parameters	• **hospi:** : OSPI handle • **cfg:** : structure that contains the memory mapped configuration information.
Return values	• **HAL:** status
Notes	• This function is used only in Memory mapped Mode

HAL_OSPI_ErrorCallback

Function name	**void HAL_OSPI_ErrorCallback (OSPI_HandleTypeDef * hospi)**
Function description	Transfer Error callback.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_AbortCpltCallback

Function name	**void HAL_OSPI_AbortCpltCallback (OSPI_HandleTypeDef * hospi)**
Function description	Abort completed callback.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_FifoThresholdCallback

Function name	**void HAL_OSPI_FifoThresholdCallback (OSPI_HandleTypeDef * hospi)**
Function description	FIFO Threshold callback.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_CmdCpltCallback

Function name	**void HAL_OSPI_CmdCpltCallback (OSPI_HandleTypeDef ***

	hospi)
Function description	Command completed callback.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_RxCpltCallback

Function name	**void HAL_OSPI_RxCpltCallback (OSPI_HandleTypeDef * hospi)**
Function description	Rx Transfer completed callback.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_TxCpltCallback

Function name	**void HAL_OSPI_TxCpltCallback (OSPI_HandleTypeDef * hospi)**
Function description	Tx Transfer completed callback.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_RxHalfCpltCallback

Function name	**void HAL_OSPI_RxHalfCpltCallback (OSPI_HandleTypeDef * hospi)**
Function description	Rx Half Transfer completed callback.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_TxHalfCpltCallback

Function name	**void HAL_OSPI_TxHalfCpltCallback (OSPI_HandleTypeDef * hospi)**
Function description	Tx Half Transfer completed callback.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_StatusMatchCallback

Function name	**void HAL_OSPI_StatusMatchCallback (OSPI_HandleTypeDef * hospi)**
Function description	Status Match callback.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_TimeOutCallback

Function name	**void HAL_OSPI_TimeOutCallback (OSPI_HandleTypeDef * hospi)**
Function description	Timeout callback.
Parameters	• **hospi:** : OSPI handle
Return values	• **None:**

HAL_OSPI_Abort

Function name	**HAL_StatusTypeDef HAL_OSPI_Abort (OSPI_HandleTypeDef * hospi)**
Function description	Abort the current transmission.
Parameters	• **hospi:** : OSPI handle
Return values	• **HAL:** status

HAL_OSPI_Abort_IT

Function name	**HAL_StatusTypeDef HAL_OSPI_Abort_IT (OSPI_HandleTypeDef * hospi)**
Function description	Abort the current transmission (non-blocking function)
Parameters	• **hospi:** : OSPI handle
Return values	• **HAL:** status

HAL_OSPI_SetFifoThreshold

Function name	**HAL_StatusTypeDef HAL_OSPI_SetFifoThreshold (OSPI_HandleTypeDef * hospi, uint32_t Threshold)**
Function description	Set OSPI Fifo threshold.
Parameters	• **hospi:** : OSPI handle. • **Threshold:** : Threshold of the Fifo.
Return values	• **HAL:** status

HAL_OSPI_GetFifoThreshold

Function name	**uint32_t HAL_OSPI_GetFifoThreshold (OSPI_HandleTypeDef * hospi)**
Function description	Get OSPI Fifo threshold.
Parameters	• **hospi:** : OSPI handle.
Return values	• **Fifo:** threshold

HAL_OSPI_SetTimeout

Function name	**HAL_StatusTypeDef HAL_OSPI_SetTimeout (OSPI_HandleTypeDef * hospi, uint32_t Timeout)**

Function description	Set OSPI timeout.
Parameters	• **hospi:** : OSPI handle. • **Timeout:** : Timeout for the memory access.
Return values	• **None:**

HAL_OSPI_GetError

Function name	**uint32_t HAL_OSPI_GetError (OSPI_HandleTypeDef * hospi)**
Function description	Return the OSPI error code.
Parameters	• **hospi:** : OSPI handle
Return values	• **OSPI:** Error Code

HAL_OSPI_GetState

Function name	**uint32_t HAL_OSPI_GetState (OSPI_HandleTypeDef * hospi)**
Function description	Return the OSPI handle state.
Parameters	• **hospi:** : OSPI handle
Return values	• **HAL:** state

HAL_OSPIM_Config

Function name	**HAL_StatusTypeDef HAL_OSPIM_Config (OSPI_HandleTypeDef * hospi, OSPIM_CfgTypeDef * cfg, uint32_t Timeout)**
Function description	Configure the OctoSPI IO manager.
Parameters	• **hospi:** : OSPI handle • **cfg:** : Configuration of the IO Manager for the instance • **Timeout:** : Timeout duration
Return values	• **HAL:** status

45.3 OSPI Firmware driver defines

45.3.1 OSPI

OSPI Address DTR Mode

HAL_OSPI_ADDRESS_DTR_DISABLE	DTR mode disabled for address phase
HAL_OSPI_ADDRESS_DTR_ENABLE	DTR mode enabled for address phase

OSPI Address Mode

HAL_OSPI_ADDRESS_NONE	No address
HAL_OSPI_ADDRESS_1_LINE	Address on a single line
HAL_OSPI_ADDRESS_2_LINES	Address on two lines
HAL_OSPI_ADDRESS_4_LINES	Address on four lines
HAL_OSPI_ADDRESS_8_LINES	Address on eight lines

OSPI Address Size

HAL_OSPI_ADDRESS_8_BITS	8-bit address
HAL_OSPI_ADDRESS_16_BITS	16-bit address
HAL_OSPI_ADDRESS_24_BITS	24-bit address
HAL_OSPI_ADDRESS_32_BITS	32-bit address

OSPI Hyperbus Address Space

HAL_OSPI_MEMORY_ADDRESS_SPACE	HyperBus memory mode
HAL_OSPI_REGISTER_ADDRESS_SPACE	HyperBus register mode

OSPI Alternate Bytes DTR Mode

HAL_OSPI_ALTERNATE_BYTES_DTR_DISABLE	DTR mode disabled for alternate bytes phase
HAL_OSPI_ALTERNATE_BYTES_DTR_ENABLE	DTR mode enabled for alternate bytes phase

OSPI Alternate Bytes Mode

HAL_OSPI_ALTERNATE_BYTES_NONE	No alternate bytes
HAL_OSPI_ALTERNATE_BYTES_1_LINE	Alternate bytes on a single line
HAL_OSPI_ALTERNATE_BYTES_2_LINES	Alternate bytes on two lines
HAL_OSPI_ALTERNATE_BYTES_4_LINES	Alternate bytes on four lines
HAL_OSPI_ALTERNATE_BYTES_8_LINES	Alternate bytes on eight lines

OSPI Alternate Bytes Size

HAL_OSPI_ALTERNATE_BYTES_8_BITS	8-bit alternate bytes
HAL_OSPI_ALTERNATE_BYTES_16_BITS	16-bit alternate bytes
HAL_OSPI_ALTERNATE_BYTES_24_BITS	24-bit alternate bytes
HAL_OSPI_ALTERNATE_BYTES_32_BITS	32-bit alternate bytes

OSPI Automatic Stop

HAL_OSPI_AUTOMATIC_STOP_DISABLE	AutoPolling stops only with abort or OSPI disabling
HAL_OSPI_AUTOMATIC_STOP_ENABLE	AutoPolling stops as soon as there is a match

OSPI Clock Mode

HAL_OSPI_CLOCK_MODE_0	CLK must stay low while nCS is high
HAL_OSPI_CLOCK_MODE_3	CLK must stay high while nCS is high

OSPI Data DTR Mode

HAL_OSPI_DATA_DTR_DISABLE	DTR mode disabled for data phase
HAL_OSPI_DATA_DTR_ENABLE	DTR mode enabled for data phase

OSPI Data Mode

HAL_OSPI_DATA_NONE	No data
HAL_OSPI_DATA_1_LINE	Data on a single line

HAL_OSPI_DATA_2_LINES	Data on two lines
HAL_OSPI_DATA_4_LINES	Data on four lines
HAL_OSPI_DATA_8_LINES	Data on eight lines

OSPI Delay Hold Quarter Cycle

HAL_OSPI_DHQC_DISABLE	No Delay
HAL_OSPI_DHQC_ENABLE	Delay Hold 1/4 cycle

OSPI DQS Mode

HAL_OSPI_DQS_DISABLE	DQS disabled
HAL_OSPI_DQS_ENABLE	DQS enabled

OSPI Dual-Quad

HAL_OSPI_DUALQUAD_DISABLE	Dual-Quad mode disabled
HAL_OSPI_DUALQUAD_ENABLE	Dual-Quad mode enabled

OSPI Error Code

HAL_OSPI_ERROR_NONE	No error
HAL_OSPI_ERROR_TIMEOUT	Timeout error
HAL_OSPI_ERROR_TRANSFER	Transfer error
HAL_OSPI_ERROR_DMA	DMA transfer error
HAL_OSPI_ERROR_INVALID_PARAM	Invalid parameters error
HAL_OSPI_ERROR_INVALID_SEQUENCE	Sequence of the state machine is incorrect

OSPI Exported Macros

__HAL_OSPI_RESET_HANDLE_STATE

Description:

- Reset OSPI handle state.

Parameters:

- __HANDLE__: OSPI handle.

Return value:

- None

__HAL_OSPI_ENABLE

Description:

- Enable the OSPI peripheral.

Parameters:

- __HANDLE__: specifies the OSPI Handle.

Return value:

- None

__HAL_OSPI_DISABLE

Description:

- Disable the OSPI peripheral.

Parameters:

- __HANDLE__: specifies the OSPI Handle.

Return value:

- None

__HAL_OSPI_ENABLE_IT

Description:

- Enable the specified OSPI interrupt.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __INTERRUPT__: specifies the OSPI interrupt source to enable. This parameter can be one of the following values:
  - HAL_OSPI_IT_TO: OSPI Timeout interrupt
  - HAL_OSPI_IT_SM: OSPI Status match interrupt
  - HAL_OSPI_IT_FT: OSPI FIFO threshold interrupt
  - HAL_OSPI_IT_TC: OSPI Transfer complete interrupt
  - HAL_OSPI_IT_TE: OSPI Transfer error interrupt

Return value:

- None

__HAL_OSPI_DISABLE_IT

Description:

- Disable the specified OSPI interrupt.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __INTERRUPT__: specifies the OSPI interrupt source to disable. This parameter can be one of the following values:
  - HAL_OSPI_IT_TO: OSPI Timeout interrupt
  - HAL_OSPI_IT_SM: OSPI Status match interrupt
  - HAL_OSPI_IT_FT: OSPI FIFO threshold interrupt
  - HAL_OSPI_IT_TC: OSPI Transfer complete interrupt
  - HAL_OSPI_IT_TE: OSPI Transfer error interrupt

Return value:

- None

__HAL_OSPI_GET_IT_SOURCE

Description:

- Check whether the specified OSPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the OSPI Handle.

- __INTERRUPT__: specifies the OSPI interrupt source to check. This parameter can be one of the following values:
  – HAL_OSPI_IT_TO: OSPI Timeout interrupt
  – HAL_OSPI_IT_SM: OSPI Status match interrupt
  – HAL_OSPI_IT_FT: OSPI FIFO threshold interrupt
  – HAL_OSPI_IT_TC: OSPI Transfer complete interrupt
  – HAL_OSPI_IT_TE: OSPI Transfer error interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_OSPI_GET_FLAG

Description:

- Check whether the selected OSPI flag is set or not.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __FLAG__: specifies the OSPI flag to check. This parameter can be one of the following values:
  – HAL_OSPI_FLAG_BUSY: OSPI Busy flag
  – HAL_OSPI_FLAG_TO: OSPI Timeout flag
  – HAL_OSPI_FLAG_SM: OSPI Status match flag
  – HAL_OSPI_FLAG_FT: OSPI FIFO threshold flag
  – HAL_OSPI_FLAG_TC: OSPI Transfer complete flag
  – HAL_OSPI_FLAG_TE: OSPI Transfer error flag

Return value:

- None

__HAL_OSPI_CLEAR_FLAG

Description:

- Clears the specified OSPI's flag status.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __FLAG__: specifies the OSPI clear register flag that needs to be set This parameter can be one of the following values:
  – HAL_OSPI_FLAG_TO: OSPI Timeout flag

- – HAL_OSPI_FLAG_SM: OSPI Status match flag
- – HAL_OSPI_FLAG_TC: OSPI Transfer complete flag
- – HAL_OSPI_FLAG_TE: OSPI Transfer error flag

Return value:

- None

OSPI Flags

HAL_OSPI_FLAG_BUSY	Busy flag: operation is ongoing
HAL_OSPI_FLAG_TO	Timeout flag: timeout occurs in memory-mapped mode
HAL_OSPI_FLAG_SM	Status match flag: received data matches in autopolling mode
HAL_OSPI_FLAG_FT	Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete
HAL_OSPI_FLAG_TC	Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted
HAL_OSPI_FLAG_TE	Transfer error flag: invalid address is being accessed

OSPI Flash Id

HAL_OSPI_FLASH_ID_1	FLASH 1 selected
HAL_OSPI_FLASH_ID_2	FLASH 2 selected

OSPI Free Running Clock

HAL_OSPI_FREERUNCLK_DISABLE	CLK is not free running
HAL_OSPI_FREERUNCLK_ENABLE	CLK is free running (always provided)

OSPI Instruction DTR Mode

HAL_OSPI_INSTRUCTION_DTR_DISABLE	DTR mode disabled for instruction phase
HAL_OSPI_INSTRUCTION_DTR_ENABLE	DTR mode enabled for instruction phase

OSPI Instruction Mode

HAL_OSPI_INSTRUCTION_NONE	No instruction
HAL_OSPI_INSTRUCTION_1_LINE	Instruction on a single line
HAL_OSPI_INSTRUCTION_2_LINES	Instruction on two lines
HAL_OSPI_INSTRUCTION_4_LINES	Instruction on four lines
HAL_OSPI_INSTRUCTION_8_LINES	Instruction on eight lines

OSPI Instruction Size

HAL_OSPI_INSTRUCTION_8_BITS	8-bit instruction
HAL_OSPI_INSTRUCTION_16_BITS	16-bit instruction
HAL_OSPI_INSTRUCTION_24_BITS	24-bit instruction
HAL_OSPI_INSTRUCTION_32_BITS	32-bit instruction

OSPI Interrupts

HAL_OSPI_IT_TO	Interrupt on the timeout flag

HAL_OSPI_IT_SM	Interrupt on the status match flag
HAL_OSPI_IT_FT	Interrupt on the fifo threshold flag
HAL_OSPI_IT_TC	Interrupt on the transfer complete flag
HAL_OSPI_IT_TE	Interrupt on the transfer error flag

OSPI Hyperbus Latency Mode

HAL_OSPI_VARIABLE_LATENCY	Variable initial latency
HAL_OSPI_FIXED_LATENCY	Fixed latency

OSPI Match Mode

HAL_OSPI_MATCH_MODE_AND	AND match mode between unmasked bits
HAL_OSPI_MATCH_MODE_OR	OR match mode between unmasked bits

OSPI Memory Type

HAL_OSPI_MEMTYPE_MICRON	Micron mode
HAL_OSPI_MEMTYPE_MACRONIX	Macronix mode
HAL_OSPI_MEMTYPE_MACRONIX_RAM	Macronix RAM mode
HAL_OSPI_MEMTYPE_HYPERBUS	Hyperbus mode

OSPI Operation Type

HAL_OSPI_OPTYPE_COMMON_CFG	Common configuration (indirect or auto-polling mode)
HAL_OSPI_OPTYPE_READ_CFG	Read configuration (memory-mapped mode)
HAL_OSPI_OPTYPE_WRITE_CFG	Write configuration (memory-mapped mode)

OSPI Sample Shifting

HAL_OSPI_SAMPLE_SHIFTING_NONE	No shift
HAL_OSPI_SAMPLE_SHIFTING_HALFCYCLE	1/2 cycle shift

OSPI SIOO Mode

HAL_OSPI_SIOO_INST_EVERY_CMD	Send instruction on every transaction
HAL_OSPI_SIOO_INST_ONLY_FIRST_CMD	Send instruction only for the first command

OSPI State

HAL_OSPI_STATE_RESET	Initial state
HAL_OSPI_STATE_HYPERBUS_INIT	Initialization done in hyperbus mode but timing configuration not done
HAL_OSPI_STATE_READY	Driver ready to be used
HAL_OSPI_STATE_CMD_CFG	Command (regular or hyperbus) configured, ready for an action
HAL_OSPI_STATE_READ_CMD_CFG	Read command configuration done, not the write command configuration
HAL_OSPI_STATE_WRITE_CMD_CFG	Write command configuration done, not the read command configuration
HAL_OSPI_STATE_BUSY_CMD	Command without data on-going

HAL_OSPI_STATE_BUSY_TX	Indirect Tx on-going
HAL_OSPI_STATE_BUSY_RX	Indirect Rx on-going
HAL_OSPI_STATE_BUSY_AUTO_POLLING	Auto-polling on-going
HAL_OSPI_STATE_BUSY_MEM_MAPPED	Memory-mapped on-going
HAL_OSPI_STATE_ABORT	Abort on-going
HAL_OSPI_STATE_ERROR	Blocking error, driver should be re-initialized

OSPI Timeout Activation

HAL_OSPI_TIMEOUT_COUNTER_DISABLE	Timeout counter disabled, nCS remains active
HAL_OSPI_TIMEOUT_COUNTER_ENABLE	Timeout counter enabled, nCS released when timeout expires

OSPI Timeout definition

HAL_OSPI_TIMEOUT_DEFAULT_VALUE

OSPI Wrap-Size

HAL_OSPI_WRAP_NOT_SUPPORTED	wrapped reads are not supported by the memory
HAL_OSPI_WRAP_16_BYTES	external memory supports wrap size of 16 bytes
HAL_OSPI_WRAP_32_BYTES	external memory supports wrap size of 32 bytes
HAL_OSPI_WRAP_64_BYTES	external memory supports wrap size of 64 bytes
HAL_OSPI_WRAP_128_BYTES	external memory supports wrap size of 128 bytes

OSPI Hyperbus Write Zero Latency Activation

HAL_OSPI_LATENCY_ON_WRITE	Latency on write accesses
HAL_OSPI_NO_LATENCY_ON_WRITE	No latency on write accesses

46 HAL PCD Generic Driver

46.1 PCD Firmware driver registers structures

46.1.1 PCD_HandleTypeDef

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***__IO uint8_t USB_Address***
- ***PCD_EPTypeDef IN_ep***
- ***PCD_EPTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***__IO PCD_StateTypeDef State***
- ***uint32_t Setup***
- ***PCD_LPM_StateTypeDef LPM_State***
- ***uint32_t BESL***
- ***uint32_t lpm_active***
- ***uint32_t battery_charging_active***
- ***void * pData***

Field Documentation

- ***PCD_TypeDef* PCD_HandleTypeDef::Instance***
  Register base address
- ***PCD_InitTypeDef PCD_HandleTypeDef::Init***
  PCD required parameters
- ***__IO uint8_t PCD_HandleTypeDef::USB_Address***
  USB Address: not used by USB OTG FS
- ***PCD_EPTypeDef PCD_HandleTypeDef::IN_ep[15]***
  IN endpoint parameters
- ***PCD_EPTypeDef PCD_HandleTypeDef::OUT_ep[15]***
  OUT endpoint parameters
- ***HAL_LockTypeDef PCD_HandleTypeDef::Lock***
  PCD peripheral status
- ***__IO PCD_StateTypeDef PCD_HandleTypeDef::State***
  PCD communication state
- ***uint32_t PCD_HandleTypeDef::Setup[12]***
  Setup packet buffer
- ***PCD_LPM_StateTypeDef PCD_HandleTypeDef::LPM_State***
  LPM State
- ***uint32_t PCD_HandleTypeDef::BESL***
- ***uint32_t PCD_HandleTypeDef::lpm_active***
  Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE
- ***uint32_t PCD_HandleTypeDef::battery_charging_active***
  Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE
- ***void* PCD_HandleTypeDef::pData***
  Pointer to upper stack Handler

46.2 PCD Firmware driver API description

46.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD_HandleTypeDef handle structure, for example: PCD_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_PCD_Init() API to initialize the PCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL_PCD_MspInit() API:
   a. Enable the PCD/USB Low Level interface clock using
      – __HAL_RCC_USB_OTG_FS_CLK_ENABLE();
   b. Initialize the related GPIO clocks
   c. Configure PCD pin-out
   d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
   a. hpcd.pData = pdev;
6. Enable PCD transmission and reception:
   a. HAL_PCD_Start();

46.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- ***HAL_PCD_Init()***
- ***HAL_PCD_DeInit()***
- ***HAL_PCD_MspInit()***
- ***HAL_PCD_MspDeInit()***

46.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PCD data transfers.

This section contains the following APIs:

- ***HAL_PCD_Start()***
- ***HAL_PCD_Stop()***
- ***HAL_PCD_IRQHandler()***
- ***HAL_PCD_DataOutStageCallback()***
- ***HAL_PCD_DataInStageCallback()***
- ***HAL_PCD_SetupStageCallback()***
- ***HAL_PCD_SOFCallback()***
- ***HAL_PCD_ResetCallback()***
- ***HAL_PCD_SuspendCallback()***
- ***HAL_PCD_ResumeCallback()***
- ***HAL_PCD_ISOOUTIncompleteCallback()***
- ***HAL_PCD_ISOINIncompleteCallback()***
- ***HAL_PCD_ConnectCallback()***
- ***HAL_PCD_DisconnectCallback()***

46.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.

This section contains the following APIs:

- *HAL_PCD_DevConnect()*
- *HAL_PCD_DevDisconnect()*
- *HAL_PCD_SetAddress()*
- *HAL_PCD_EP_Open()*
- *HAL_PCD_EP_Close()*
- *HAL_PCD_EP_Receive()*
- *HAL_PCD_EP_GetRxCount()*
- *HAL_PCD_EP_Transmit()*
- *HAL_PCD_EP_SetStall()*
- *HAL_PCD_EP_ClrStall()*
- *HAL_PCD_EP_Flush()*
- *HAL_PCD_ActivateRemoteWakeup()*
- *HAL_PCD_DeActivateRemoteWakeup()*

46.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_PCD_GetState()*

46.2.6 Detailed description of functions

HAL_PCD_Init

Function name	**HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)**
Function description	Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and initialize the associated handle.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_DeInit

Function name	**HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)**
Function description	DeInitializes the PCD peripheral.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_MspInit

Function name	**void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)**
Function description	Initializes the PCD MSP.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_MspDeInit

Function name	**void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)**
Function description	DeInitializes PCD MSP.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_Start

Function name	**HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)**
Function description	Start The USB OTG Device.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_Stop

Function name	**HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)**
Function description	Stop The USB OTG Device.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_IRQHandler

Function name	**void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)**
Function description	Handles PCD interrupt request.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_DataOutStageCallback

Function name	**void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function description	Data OUT stage callback.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number
Return values	• **None:**

HAL_PCD_DataInStageCallback

Function name	**void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function description	Data IN stage callback.
Parameters	• **hpcd:** PCD handle

- **epnum:** endpoint number

Return values
- **None:**

HAL_PCD_SetupStageCallback

Function name	**void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)**
Function description	Setup stage callback.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_SOFCallback

Function name	**void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)**
Function description	USB Start Of Frame callback.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_ResetCallback

Function name	**void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)**
Function description	USB Reset callback.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_SuspendCallback

Function name	**void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)**
Function description	Suspend event callback.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_ResumeCallback

Function name	**void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)**
Function description	Resume event callback.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_ISOOUTIncompleteCallback

Function name	**void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**

Function description	Incomplete ISO OUT callback.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number
Return values	• **None:**

HAL_PCD_ISOINIncompleteCallback

Function name	**void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function description	Incomplete ISO IN callback.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number
Return values	• **None:**

HAL_PCD_ConnectCallback

Function name	**void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)**
Function description	Connection event callback.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_DisconnectCallback

Function name	**void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)**
Function description	Disconnection event callback.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_DevConnect

Function name	**HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)**
Function description	Connect the USB device.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_DevDisconnect

Function name	**HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)**
Function description	Disconnect the USB device.
Parameters	• **hpcd:** PCD handle

Return values
- **HAL:** status

HAL_PCD_SetAddress

Function name
HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)

Function description
Set the USB Device address.

Parameters
- **hpcd:** PCD handle
- **address:** new device address

Return values
- **HAL:** status

HAL_PCD_EP_Open

Function name
HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)

Function description
Open and configure an endpoint.

Parameters
- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **ep_mps:** endpoint max packet size
- **ep_type:** endpoint type

Return values
- **HAL:** status

HAL_PCD_EP_Close

Function name
HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description
Deactivate an endpoint.

Parameters
- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values
- **HAL:** status

HAL_PCD_EP_Receive

Function name
HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description
Receive an amount of data.

Parameters
- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the reception buffer
- **len:** amount of data to be received

Return values
- **HAL:** status

HAL_PCD_EP_Transmit

Function name
HAL_StatusTypeDef HAL_PCD_EP_Transmit

	(PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)
Function description	Send an amount of data.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address • **pBuf:** pointer to the transmission buffer • **len:** amount of data to be sent
Return values	• **HAL:** status

HAL_PCD_EP_GetRxCount

Function name	**uint16_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Get Received Data Size.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **Data:** Size

HAL_PCD_EP_SetStall

Function name	**HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Set a STALL condition over an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_EP_ClrStall

Function name	**HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Clear a STALL condition over in an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_EP_Flush

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Flush an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_ActivateRemoteWakeup

Function name	**HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)**
Function description	Activate remote wakeup signalling.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_DeActivateRemoteWakeup

Function name	**HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)**
Function description	De-activate remote wakeup signalling.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_GetState

Function name	**PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)**
Function description	Return the PCD handle state.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** state

46.3 PCD Firmware driver defines

46.3.1 PCD

PCD Exported Macros

__HAL_PCD_ENABLE

__HAL_PCD_DISABLE

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

__HAL_PCD_IS_INVALID_INTERRUPT

__HAL_PCD_UNGATE_PHYCLOCK

__HAL_PCD_GATE_PHYCLOCK

__HAL_PCD_IS_PHY_SUSPENDED

__HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_IT

__HAL_USB_OTG_FS_WAKEUP_EXTI_DISABLE_IT

__HAL_USB_OTG_FS_WAKEUP_EXTI_GET_FLAG

__HAL_USB_OTG_FS_WAKEUP_EXTI_CLEAR_FLAG

__HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_RISING_EDGE

__HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_FALLING_EDGE

__HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE

__HAL_USB_OTG_FS_WAKEUP_EXTI_GENERATE_SWIT

PCD PHY Module

PCD_PHY_EMBEDDED

PCD Speed

PCD_SPEED_FULL

Turnaround Timeout Value

USBD_FS_TRDT_VALUE

47 HAL PCD Extension Driver

47.1 PCDEx Firmware driver API description

47.1.1 Extended features functions

This section provides functions allowing to:

- Update FIFO configuration

This section contains the following APIs:

- *HAL_PCDEx_SetTxFiFo()*
- *HAL_PCDEx_SetRxFiFo()*
- *HAL_PCDEx_ActivateLPM()*
- *HAL_PCDEx_DeActivateLPM()*
- *HAL_PCDEx_BCD_VBUSDetect()*
- *HAL_PCDEx_ActivateBCD()*
- *HAL_PCDEx_DeActivateBCD()*
- *HAL_PCDEx_LPM_Callback()*
- *HAL_PCDEx_BCD_Callback()*

47.1.2 Detailed description of functions

HAL_PCDEx_SetTxFiFo

Function name	**HAL_StatusTypeDef HAL_PCDEx_SetTxFiFo (PCD_HandleTypeDef * hpcd, uint8_t fifo, uint16_t size)**
Function description	Set Tx FIFO.
Parameters	• **hpcd:** PCD handle • **fifo:** The number of Tx fifo • **size:** Fifo size
Return values	• **HAL:** status

HAL_PCDEx_SetRxFiFo

Function name	**HAL_StatusTypeDef HAL_PCDEx_SetRxFiFo (PCD_HandleTypeDef * hpcd, uint16_t size)**
Function description	Set Rx FIFO.
Parameters	• **hpcd:** PCD handle • **size:** Size of Rx fifo
Return values	• **HAL:** status

HAL_PCDEx_ActivateLPM

Function name	**HAL_StatusTypeDef HAL_PCDEx_ActivateLPM (PCD_HandleTypeDef * hpcd)**
Function description	Activate LPM feature.

Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCDEx_DeActivateLPM

Function name	**HAL_StatusTypeDef HAL_PCDEx_DeActivateLPM (PCD_HandleTypeDef * hpcd)**
Function description	Deactivate LPM feature.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCDEx_ActivateBCD

Function name	**HAL_StatusTypeDef HAL_PCDEx_ActivateBCD (PCD_HandleTypeDef * hpcd)**
Function description	Activate BatteryCharging feature.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCDEx_DeActivateBCD

Function name	**HAL_StatusTypeDef HAL_PCDEx_DeActivateBCD (PCD_HandleTypeDef * hpcd)**
Function description	Deactivate BatteryCharging feature.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCDEx_BCD_VBUSDetect

Function name	**void HAL_PCDEx_BCD_VBUSDetect (PCD_HandleTypeDef * hpcd)**
Function description	Handle BatteryCharging Process.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCDEx_LPM_Callback

Function name	**void HAL_PCDEx_LPM_Callback (PCD_HandleTypeDef * hpcd, PCD_LPM_MsgTypeDef msg)**
Function description	Send LPM message to user layer callback.
Parameters	• **hpcd:** PCD handle • **msg:** LPM message
Return values	• **HAL:** status

HAL_PCDEx_BCD_Callback

Function name	**void HAL_PCDEx_BCD_Callback (PCD_HandleTypeDef * hpcd, PCD_BCD_MsgTypeDef msg)**
Function description	Send BatteryCharging message to user layer callback.
Parameters	• **hpcd:** PCD handle • **msg:** LPM message
Return values	• **HAL:** status

48 HAL QSPI Generic Driver

48.1 QSPI Firmware driver registers structures

48.1.1 QSPI_InitTypeDef

Data Fields

- *uint32_t ClockPrescaler*
- *uint32_t FifoThreshold*
- *uint32_t SampleShifting*
- *uint32_t FlashSize*
- *uint32_t ChipSelectHighTime*
- *uint32_t ClockMode*
- *uint32_t FlashID*
- *uint32_t DualFlash*

Field Documentation

- *uint32_t QSPI_InitTypeDef::ClockPrescaler*
- *uint32_t QSPI_InitTypeDef::FifoThreshold*
- *uint32_t QSPI_InitTypeDef::SampleShifting*
- *uint32_t QSPI_InitTypeDef::FlashSize*
- *uint32_t QSPI_InitTypeDef::ChipSelectHighTime*
- *uint32_t QSPI_InitTypeDef::ClockMode*
- *uint32_t QSPI_InitTypeDef::FlashID*
- *uint32_t QSPI_InitTypeDef::DualFlash*

48.1.2 QSPI_HandleTypeDef

Data Fields

- *QUADSPI_TypeDef * Instance*
- *QSPI_InitTypeDef Init*
- *uint8_t * pTxBuffPtr*
- *__IO uint32_t TxXferSize*
- *__IO uint32_t TxXferCount*
- *uint8_t * pRxBuffPtr*
- *__IO uint32_t RxXferSize*
- *__IO uint32_t RxXferCount*
- *DMA_HandleTypeDef * hdma*
- *__IO HAL_LockTypeDef Lock*
- *__IO HAL_QSPI_StateTypeDef State*
- *__IO uint32_t ErrorCode*
- *uint32_t Timeout*

Field Documentation

- *QUADSPI_TypeDef* QSPI_HandleTypeDef::Instance*
- *QSPI_InitTypeDef QSPI_HandleTypeDef::Init*
- *uint8_t* QSPI_HandleTypeDef::pTxBuffPtr*
- *__IO uint32_t QSPI_HandleTypeDef::TxXferSize*
- *__IO uint32_t QSPI_HandleTypeDef::TxXferCount*
- *uint8_t* QSPI_HandleTypeDef::pRxBuffPtr*

- ***__IO uint32_t QSPI_HandleTypeDef::RxXferSize***
- ***__IO uint32_t QSPI_HandleTypeDef::RxXferCount***
- ***DMA_HandleTypeDef* QSPI_HandleTypeDef::hdma***
- ***__IO HAL_LockTypeDef QSPI_HandleTypeDef::Lock***
- ***__IO HAL_QSPI_StateTypeDef QSPI_HandleTypeDef::State***
- ***__IO uint32_t QSPI_HandleTypeDef::ErrorCode***
- ***uint32_t QSPI_HandleTypeDef::Timeout***

48.1.3 QSPI_CommandTypeDef

Data Fields

- ***uint32_t Instruction***
- ***uint32_t Address***
- ***uint32_t AlternateBytes***
- ***uint32_t AddressSize***
- ***uint32_t AlternateBytesSize***
- ***uint32_t DummyCycles***
- ***uint32_t InstructionMode***
- ***uint32_t AddressMode***
- ***uint32_t AlternateByteMode***
- ***uint32_t DataMode***
- ***uint32_t NbData***
- ***uint32_t DdrMode***
- ***uint32_t DdrHoldHalfCycle***
- ***uint32_t SIOOMode***

Field Documentation

- ***uint32_t QSPI_CommandTypeDef::Instruction***
- ***uint32_t QSPI_CommandTypeDef::Address***
- ***uint32_t QSPI_CommandTypeDef::AlternateBytes***
- ***uint32_t QSPI_CommandTypeDef::AddressSize***
- ***uint32_t QSPI_CommandTypeDef::AlternateBytesSize***
- ***uint32_t QSPI_CommandTypeDef::DummyCycles***
- ***uint32_t QSPI_CommandTypeDef::InstructionMode***
- ***uint32_t QSPI_CommandTypeDef::AddressMode***
- ***uint32_t QSPI_CommandTypeDef::AlternateByteMode***
- ***uint32_t QSPI_CommandTypeDef::DataMode***
- ***uint32_t QSPI_CommandTypeDef::NbData***
- ***uint32_t QSPI_CommandTypeDef::DdrMode***
- ***uint32_t QSPI_CommandTypeDef::DdrHoldHalfCycle***
- ***uint32_t QSPI_CommandTypeDef::SIOOMode***

48.1.4 QSPI_AutoPollingTypeDef

Data Fields

- ***uint32_t Match***
- ***uint32_t Mask***
- ***uint32_t Interval***
- ***uint32_t StatusBytesSize***
- ***uint32_t MatchMode***
- ***uint32_t AutomaticStop***

Field Documentation

- ***uint32_t QSPI_AutoPollingTypeDef::Match***
- ***uint32_t QSPI_AutoPollingTypeDef::Mask***
- ***uint32_t QSPI_AutoPollingTypeDef::Interval***
- ***uint32_t QSPI_AutoPollingTypeDef::StatusBytesSize***
- ***uint32_t QSPI_AutoPollingTypeDef::MatchMode***
- ***uint32_t QSPI_AutoPollingTypeDef::AutomaticStop***

48.1.5 QSPI_MemoryMappedTypeDef

Data Fields

- ***uint32_t TimeOutPeriod***
- ***uint32_t TimeOutActivation***

Field Documentation

- ***uint32_t QSPI_MemoryMappedTypeDef::TimeOutPeriod***
- ***uint32_t QSPI_MemoryMappedTypeDef::TimeOutActivation***

48.2 QSPI Firmware driver API description

48.2.1 How to use this driver

Initialization

1. As prerequisite, fill in the HAL_QSPI_MspInit():
   – Enable QuadSPI clock interface with __HAL_RCC_QSPI_CLK_ENABLE().
   – Reset QuadSPI IP with __HAL_RCC_QSPI_FORCE_RESET() and __HAL_RCC_QSPI_RELEASE_RESET().
   – Enable the clocks for the QuadSPI GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
   – Configure these QuadSPI pins in alternate mode using HAL_GPIO_Init().
   – If interrupt mode is used, enable and configure QuadSPI global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
   – If DMA mode is used, enable the clocks for the QuadSPI DMA channel with __HAL_RCC_DMAx_CLK_ENABLE(), configure DMA with HAL_DMA_Init(), link it with QuadSPI handle using __HAL_LINKDMA(), enable and configure DMA channel global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Configure the flash size, the clock prescaler, the fifo threshold, the clock mode, the sample shifting and the CS high time using the HAL_QSPI_Init() function.

Indirect functional mode

1. Configure the command sequence using the HAL_QSPI_Command() or HAL_QSPI_Command_IT() functions:
   – Instruction phase: the mode used and if present the instruction opcode.
   – Address phase: the mode used and if present the size and the address value.
   – Alternate-bytes phase: the mode used and if present the size and the alternate bytes values.
   – Dummy-cycles phase: the number of dummy cycles (mode used is same as data phase).
   – Data phase: the mode used and if present the number of bytes.

- Double Data Rate (DDR) mode: the activation (or not) of this mode and the delay if activated.
- Sending Instruction Only Once (SIOO) mode: the activation (or not) of this mode.

2. If no data is required for the command, it is sent directly to the memory:
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_QSPI_CmdCpltCallback() will be called when the transfer is complete.
3. For the indirect write mode, use HAL_QSPI_Transmit(), HAL_QSPI_Transmit_DMA() or HAL_QSPI_Transmit_IT() after the command configuration:
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_QSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_QSPI_TxCpltCallback() will be called when the transfer is complete.
   - In DMA mode, HAL_QSPI_TxHalfCpltCallback() will be called at the half transfer and HAL_QSPI_TxCpltCallback() will be called when the transfer is complete.
4. For the indirect read mode, use HAL_QSPI_Receive(), HAL_QSPI_Receive_DMA() or HAL_QSPI_Receive_IT() after the command configuration:
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_QSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_QSPI_RxCpltCallback() will be called when the transfer is complete.
   - In DMA mode, HAL_QSPI_RxHalfCpltCallback() will be called at the half transfer and HAL_QSPI_RxCpltCallback() will be called when the transfer is complete.

Auto-polling functional mode

1. Configure the command sequence and the auto-polling functional mode using the HAL_QSPI_AutoPolling() or HAL_QSPI_AutoPolling_IT() functions:
   - Instruction phase: the mode used and if present the instruction opcode.
   - Address phase: the mode used and if present the size and the address value.
   - Alternate-bytes phase: the mode used and if present the size and the alternate bytes values.
   - Dummy-cycles phase: the number of dummy cycles (mode used is same as data phase).
   - Data phase: the mode used.
   - Double Data Rate (DDR) mode: the activation (or not) of this mode and the delay if activated.
   - Sending Instruction Only Once (SIOO) mode: the activation (or not) of this mode.
   - The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.
2. After the configuration:
   - In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
   - In interrupt mode, HAL_QSPI_StatusMatchCallback() will be called each time the status match is reached.

Memory-mapped functional mode

1. Configure the command sequence and the memory-mapped functional mode using the HAL_QSPI_MemoryMapped() functions:
   - Instruction phase: the mode used and if present the instruction opcode.
   - Address phase: the mode used and the size.
   - Alternate-bytes phase: the mode used and if present the size and the alternate bytes values.

   – Dummy-cycles phase: the number of dummy cycles (mode used is same as data phase).
   – Data phase: the mode used.
   – Double Data Rate (DDR) mode: the activation (or not) of this mode and the delay if activated.
   – Sending Instruction Only Once (SIOO) mode: the activation (or not) of this mode.
   – The timeout activation and the timeout period.
2. After the configuration, the QuadSPI will be used as soon as an access on the AHB is done on the address range. HAL_QSPI_TimeOutCallback() will be called when the timeout expires.

Errors management and abort functionality

1. HAL_QSPI_GetError() function gives the error raised during the last operation.
2. HAL_QSPI_Abort() and HAL_QSPI_AbortIT() functions aborts any on-going operation and flushes the fifo:
   – In polling mode, the output of the function is done when the transfer complete bit is set and the busy bit cleared.
   – In interrupt mode, HAL_QSPI_AbortCpltCallback() will be called when the transfer complete bi is set.

Control functions

1. HAL_QSPI_GetState() function gives the current state of the HAL QuadSPI driver.
2. HAL_QSPI_SetTimeout() function configures the timeout value used in the driver.
3. HAL_QSPI_SetFifoThreshold() function configures the threshold on the Fifo of the QSPI IP.
4. HAL_QSPI_GetFifoThreshold() function gives the current of the Fifo's threshold

Workarounds linked to Silicon Limitation

1. Workarounds Implemented inside HAL Driver
   – Extra data written in the FIFO at the end of a read transfer

48.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to:

- Initialize the QuadSPI.
- De-initialize the QuadSPI.

This section contains the following APIs:

- ***HAL_QSPI_Init()***
- ***HAL_QSPI_DeInit()***
- ***HAL_QSPI_MspInit()***
- ***HAL_QSPI_MspDeInit()***

48.2.3 IO operation functions

This subsection provides a set of functions allowing to:

- Handle the interrupts.
- Handle the command sequence.
- Transmit data in blocking, interrupt or DMA mode.
- Receive data in blocking, interrupt or DMA mode.
- Manage the auto-polling functional mode.
- Manage the memory-mapped functional mode.

This section contains the following APIs:

- *HAL_QSPI_IRQHandler()*
- *HAL_QSPI_Command()*
- *HAL_QSPI_Command_IT()*
- *HAL_QSPI_Transmit()*
- *HAL_QSPI_Receive()*
- *HAL_QSPI_Transmit_IT()*
- *HAL_QSPI_Receive_IT()*
- *HAL_QSPI_Transmit_DMA()*
- *HAL_QSPI_Receive_DMA()*
- *HAL_QSPI_AutoPolling()*
- *HAL_QSPI_AutoPolling_IT()*
- *HAL_QSPI_MemoryMapped()*
- *HAL_QSPI_ErrorCallback()*
- *HAL_QSPI_AbortCpltCallback()*
- *HAL_QSPI_CmdCpltCallback()*
- *HAL_QSPI_RxCpltCallback()*
- *HAL_QSPI_TxCpltCallback()*
- *HAL_QSPI_RxHalfCpltCallback()*
- *HAL_QSPI_TxHalfCpltCallback()*
- *HAL_QSPI_FifoThresholdCallback()*
- *HAL_QSPI_StatusMatchCallback()*
- *HAL_QSPI_TimeOutCallback()*

48.2.4 Peripheral Control and State functions

This subsection provides a set of functions allowing to:

- Check in run-time the state of the driver.
- Check the error code set during last operation.
- Abort any operation.

This section contains the following APIs:

- *HAL_QSPI_GetState()*
- *HAL_QSPI_GetError()*
- *HAL_QSPI_Abort()*
- *HAL_QSPI_Abort_IT()*
- *HAL_QSPI_SetTimeout()*
- *HAL_QSPI_SetFifoThreshold()*
- *HAL_QSPI_GetFifoThreshold()*

48.2.5 Detailed description of functions

HAL_QSPI_Init

Function name	**HAL_StatusTypeDef HAL_QSPI_Init (QSPI_HandleTypeDef * hqspi)**
Function description	Initialize the QSPI mode according to the specified parameters in the QSPI_InitTypeDef and initialize the associated handle.
Parameters	• **hqspi:** QSPI handle
Return values	• **HAL:** status

HAL_QSPI_DeInit

Function name	**HAL_StatusTypeDef HAL_QSPI_DeInit (QSPI_HandleTypeDef * hqspi)**
Function description	De-Initialize the QSPI peripheral.
Parameters	• **hqspi:** QSPI handle
Return values	• **HAL:** status

HAL_QSPI_MspInit

Function name	**void HAL_QSPI_MspInit (QSPI_HandleTypeDef * hqspi)**
Function description	Initialize the QSPI MSP.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_MspDeInit

Function name	**void HAL_QSPI_MspDeInit (QSPI_HandleTypeDef * hqspi)**
Function description	DeInitialize the QSPI MSP.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_IRQHandler

Function name	**void HAL_QSPI_IRQHandler (QSPI_HandleTypeDef * hqspi)**
Function description	Handle QSPI interrupt request.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_Command

Function name	**HAL_StatusTypeDef HAL_QSPI_Command (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, uint32_t Timeout)**
Function description	Set the command configuration.
Parameters	• **hqspi:** QSPI handle • **cmd:** : structure that contains the command configuration information • **Timeout:** : Timeout duration
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Read or Write Modes

HAL_QSPI_Transmit

Function name	**HAL_StatusTypeDef HAL_QSPI_Transmit (QSPI_HandleTypeDef * hqspi, uint8_t * pData, uint32_t**

	Timeout)
Function description	Transmit an amount of data in blocking mode.
Parameters	• **hqspi:** QSPI handle • **pData:** pointer to data buffer • **Timeout:** : Timeout duration
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Write Mode

HAL_QSPI_Receive

Function name	**HAL_StatusTypeDef HAL_QSPI_Receive (QSPI_HandleTypeDef * hqspi, uint8_t * pData, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **hqspi:** QSPI handle • **pData:** pointer to data buffer • **Timeout:** : Timeout duration
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Read Mode

HAL_QSPI_Command_IT

Function name	**HAL_StatusTypeDef HAL_QSPI_Command_IT (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd)**
Function description	Set the command configuration in interrupt mode.
Parameters	• **hqspi:** QSPI handle • **cmd:** : structure that contains the command configuration information
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Read or Write Modes

HAL_QSPI_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_QSPI_Transmit_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)**
Function description	Send an amount of data in non-blocking mode with interrupt.
Parameters	• **hqspi:** QSPI handle • **pData:** pointer to data buffer
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Write Mode

HAL_QSPI_Receive_IT

Function name	**HAL_StatusTypeDef HAL_QSPI_Receive_IT**

	(QSPI_HandleTypeDef * hqspi, uint8_t * pData)
Function description	Receive an amount of data in non-blocking mode with interrupt.
Parameters	• **hqspi:** QSPI handle • **pData:** pointer to data buffer
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Read Mode

HAL_QSPI_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_QSPI_Transmit_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)**
Function description	Send an amount of data in non-blocking mode with DMA.
Parameters	• **hqspi:** QSPI handle • **pData:** pointer to data buffer
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Write Mode • If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword • If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_QSPI_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_QSPI_Receive_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)**
Function description	Receive an amount of data in non-blocking mode with DMA.
Parameters	• **hqspi:** QSPI handle • **pData:** pointer to data buffer.
Return values	• **HAL:** status
Notes	• This function is used only in Indirect Read Mode • If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword • If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_QSPI_AutoPolling

Function name	**HAL_StatusTypeDef HAL_QSPI_AutoPolling (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg, uint32_t Timeout)**
Function description	Configure the QSPI Automatic Polling Mode in blocking mode.
Parameters	• **hqspi:** QSPI handle • **cmd:** structure that contains the command configuration information.

- **cfg:** structure that contains the polling configuration information.
- **Timeout:** : Timeout duration

Return values
- **HAL:** status

Notes
- This function is used only in Automatic Polling Mode

HAL_QSPI_AutoPolling_IT

Function name	**HAL_StatusTypeDef HAL_QSPI_AutoPolling_IT (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg)**
Function description	Configure the QSPI Automatic Polling Mode in non-blocking mode.
Parameters	• **hqspi:** QSPI handle • **cmd:** structure that contains the command configuration information. • **cfg:** structure that contains the polling configuration information.
Return values	• **HAL:** status
Notes	• This function is used only in Automatic Polling Mode

HAL_QSPI_MemoryMapped

Function name	**HAL_StatusTypeDef HAL_QSPI_MemoryMapped (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_MemoryMappedTypeDef * cfg)**
Function description	Configure the Memory Mapped mode.
Parameters	• **hqspi:** QSPI handle • **cmd:** structure that contains the command configuration information. • **cfg:** structure that contains the memory mapped configuration information.
Return values	• **HAL:** status
Notes	• This function is used only in Memory mapped Mode

HAL_QSPI_ErrorCallback

Function name	**void HAL_QSPI_ErrorCallback (QSPI_HandleTypeDef * hqspi)**
Function description	Transfer Error callback.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_AbortCpltCallback

Function name	**void HAL_QSPI_AbortCpltCallback (QSPI_HandleTypeDef * hqspi)**
Function description	Abort completed callback.

Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_FifoThresholdCallback

Function name	**void HAL_QSPI_FifoThresholdCallback (QSPI_HandleTypeDef * hqspi)**
Function description	FIFO Threshold callback.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_CmdCpltCallback

Function name	**void HAL_QSPI_CmdCpltCallback (QSPI_HandleTypeDef * hqspi)**
Function description	Command completed callback.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_RxCpltCallback

Function name	**void HAL_QSPI_RxCpltCallback (QSPI_HandleTypeDef * hqspi)**
Function description	Rx Transfer completed callback.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_TxCpltCallback

Function name	**void HAL_QSPI_TxCpltCallback (QSPI_HandleTypeDef * hqspi)**
Function description	Tx Transfer completed callback.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_RxHalfCpltCallback

Function name	**void HAL_QSPI_RxHalfCpltCallback (QSPI_HandleTypeDef * hqspi)**
Function description	Rx Half Transfer completed callback.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_TxHalfCpltCallback

Function name	**void HAL_QSPI_TxHalfCpltCallback (QSPI_HandleTypeDef * hqspi)**
Function description	Tx Half Transfer completed callback.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_StatusMatchCallback

Function name	**void HAL_QSPI_StatusMatchCallback (QSPI_HandleTypeDef * hqspi)**
Function description	Status Match callback.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_TimeOutCallback

Function name	**void HAL_QSPI_TimeOutCallback (QSPI_HandleTypeDef * hqspi)**
Function description	Timeout callback.
Parameters	• **hqspi:** QSPI handle
Return values	• **None:**

HAL_QSPI_GetState

Function name	**HAL_QSPI_StateTypeDef HAL_QSPI_GetState (QSPI_HandleTypeDef * hqspi)**
Function description	Return the QSPI handle state.
Parameters	• **hqspi:** QSPI handle
Return values	• **HAL:** state

HAL_QSPI_GetError

Function name	**uint32_t HAL_QSPI_GetError (QSPI_HandleTypeDef * hqspi)**
Function description	Return the QSPI error code.
Parameters	• **hqspi:** QSPI handle
Return values	• **QSPI:** Error Code

HAL_QSPI_Abort

Function name	**HAL_StatusTypeDef HAL_QSPI_Abort (QSPI_HandleTypeDef * hqspi)**
Function description	Abort the current transmission.
Parameters	• **hqspi:** QSPI handle

Return values	• **HAL:** status

HAL_QSPI_Abort_IT

Function name	**HAL_StatusTypeDef HAL_QSPI_Abort_IT (QSPI_HandleTypeDef * hqspi)**
Function description	Abort the current transmission (non-blocking function)
Parameters	• **hqspi:** QSPI handle
Return values	• **HAL:** status

HAL_QSPI_SetTimeout

Function name	**void HAL_QSPI_SetTimeout (QSPI_HandleTypeDef * hqspi, uint32_t Timeout)**
Function description	Set QSPI timeout.
Parameters	• **hqspi:** QSPI handle. • **Timeout:** Timeout for the QSPI memory access.
Return values	• **None:**

HAL_QSPI_SetFifoThreshold

Function name	**HAL_StatusTypeDef HAL_QSPI_SetFifoThreshold (QSPI_HandleTypeDef * hqspi, uint32_t Threshold)**
Function description	Set QSPI Fifo threshold.
Parameters	• **hqspi:** QSPI handle. • **Threshold:** Threshold of the Fifo (value between 1 and 16).
Return values	• **HAL:** status

HAL_QSPI_GetFifoThreshold

Function name	**uint32_t HAL_QSPI_GetFifoThreshold (QSPI_HandleTypeDef * hqspi)**
Function description	Get QSPI Fifo threshold.
Parameters	• **hqspi:** QSPI handle.
Return values	• **Fifo:** threshold (value between 1 and 16)

48.3 QSPI Firmware driver defines

48.3.1 QSPI

QSPI Address Mode

QSPI_ADDRESS_NONE	No address
QSPI_ADDRESS_1_LINE	Address on a single line
QSPI_ADDRESS_2_LINES	Address on two lines
QSPI_ADDRESS_4_LINES	Address on four lines

QSPI Address Size

QSPI_ADDRESS_8_BITS	8-bit address
QSPI_ADDRESS_16_BITS	16-bit address
QSPI_ADDRESS_24_BITS	24-bit address
QSPI_ADDRESS_32_BITS	32-bit address

QSPI Alternate Bytes Mode

QSPI_ALTERNATE_BYTES_NONE	No alternate bytes
QSPI_ALTERNATE_BYTES_1_LINE	Alternate bytes on a single line
QSPI_ALTERNATE_BYTES_2_LINES	Alternate bytes on two lines
QSPI_ALTERNATE_BYTES_4_LINES	Alternate bytes on four lines

QSPI Alternate Bytes Size

QSPI_ALTERNATE_BYTES_8_BITS	8-bit alternate bytes
QSPI_ALTERNATE_BYTES_16_BITS	16-bit alternate bytes
QSPI_ALTERNATE_BYTES_24_BITS	24-bit alternate bytes
QSPI_ALTERNATE_BYTES_32_BITS	32-bit alternate bytes

QSPI Automatic Stop

QSPI_AUTOMATIC_STOP_DISABLE	AutoPolling stops only with abort or QSPI disabling
QSPI_AUTOMATIC_STOP_ENABLE	AutoPolling stops as soon as there is a match

QSPI ChipSelect High Time

QSPI_CS_HIGH_TIME_1_CYCLE	nCS stay high for at least 1 clock cycle between commands
QSPI_CS_HIGH_TIME_2_CYCLE	nCS stay high for at least 2 clock cycles between commands
QSPI_CS_HIGH_TIME_3_CYCLE	nCS stay high for at least 3 clock cycles between commands
QSPI_CS_HIGH_TIME_4_CYCLE	nCS stay high for at least 4 clock cycles between commands
QSPI_CS_HIGH_TIME_5_CYCLE	nCS stay high for at least 5 clock cycles between commands
QSPI_CS_HIGH_TIME_6_CYCLE	nCS stay high for at least 6 clock cycles between commands
QSPI_CS_HIGH_TIME_7_CYCLE	nCS stay high for at least 7 clock cycles between commands
QSPI_CS_HIGH_TIME_8_CYCLE	nCS stay high for at least 8 clock cycles between commands

QSPI Clock Mode

QSPI_CLOCK_MODE_0	Clk stays low while nCS is released
QSPI_CLOCK_MODE_3	Clk goes high while nCS is released

QSPI Data Mode

QSPI_DATA_NONE	No data
QSPI_DATA_1_LINE	Data on a single line
QSPI_DATA_2_LINES	Data on two lines
QSPI_DATA_4_LINES	Data on four lines

QSPI DDR Data Output Delay

QSPI_DDR_HHC_ANALOG_DELAY	Delay the data output using analog delay in DDR mode
QSPI_DDR_HHC_HALF_CLK_DELAY	Delay the data output by 1/2 clock cycle in DDR mode

QSPI DDR Mode

QSPI_DDR_MODE_DISABLE	Double data rate mode disabled
QSPI_DDR_MODE_ENABLE	Double data rate mode enabled

QSPI Dual Flash Mode

QSPI_DUALFLASH_ENABLE	Dual-flash mode enabled
QSPI_DUALFLASH_DISABLE	Dual-flash mode disabled

QSPI Error Code

HAL_QSPI_ERROR_NONE	No error
HAL_QSPI_ERROR_TIMEOUT	Timeout error
HAL_QSPI_ERROR_TRANSFER	Transfer error
HAL_QSPI_ERROR_DMA	DMA transfer error
HAL_QSPI_ERROR_INVALID_PARAM	Invalid parameters error

QSPI Exported Macros

__HAL_QSPI_RESET_HANDLE_STATE

Description:

- Reset QSPI handle state.

Parameters:

- __HANDLE__: QSPI handle.

Return value:

- None

__HAL_QSPI_ENABLE

Description:

- Enable the QSPI peripheral.

Parameters:

- __HANDLE__: specifies the QSPI Handle.

Return value:

- None

__HAL_QSPI_DISABLE

Description:

- Disable the QSPI peripheral.

Parameters:

- __HANDLE__: specifies the QSPI Handle.

Return value:

- None

__HAL_QSPI_ENABLE_IT

Description:

- Enable the specified QSPI interrupt.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __INTERRUPT__: specifies the QSPI interrupt source to enable. This parameter can be one of the following values:
  - QSPI_IT_TO: QSPI Timeout interrupt
  - QSPI_IT_SM: QSPI Status match interrupt
  - QSPI_IT_FT: QSPI FIFO threshold interrupt
  - QSPI_IT_TC: QSPI Transfer complete interrupt
  - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- None

__HAL_QSPI_DISABLE_IT

Description:

- Disable the specified QSPI interrupt.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __INTERRUPT__: specifies the QSPI interrupt source to disable. This parameter can be one of the following values:
  - QSPI_IT_TO: QSPI Timeout interrupt
  - QSPI_IT_SM: QSPI Status match interrupt
  - QSPI_IT_FT: QSPI FIFO threshold interrupt
  - QSPI_IT_TC: QSPI Transfer complete interrupt
  - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- None

__HAL_QSPI_GET_IT_SOURCE

Description:

- Check whether the specified QSPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __INTERRUPT__: specifies the QSPI

interrupt source to check. This parameter can be one of the following values:
  - QSPI_IT_TO: QSPI Timeout interrupt
  - QSPI_IT_SM: QSPI Status match interrupt
  - QSPI_IT_FT: QSPI FIFO threshold interrupt
  - QSPI_IT_TC: QSPI Transfer complete interrupt
  - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_QSPI_GET_FLAG

Description:

- Check whether the selected QSPI flag is set or not.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __FLAG__: specifies the QSPI flag to check. This parameter can be one of the following values:
  - QSPI_FLAG_BUSY: QSPI Busy flag
  - QSPI_FLAG_TO: QSPI Timeout flag
  - QSPI_FLAG_SM: QSPI Status match flag
  - QSPI_FLAG_FT: QSPI FIFO threshold flag
  - QSPI_FLAG_TC: QSPI Transfer complete flag
  - QSPI_FLAG_TE: QSPI Transfer error flag

Return value:

- None

__HAL_QSPI_CLEAR_FLAG

Description:

- Clears the specified QSPI's flag status.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __FLAG__: specifies the QSPI clear register flag that needs to be set This parameter can be one of the following values:
  - QSPI_FLAG_TO: QSPI Timeout flag
  - QSPI_FLAG_SM: QSPI Status match flag
  - QSPI_FLAG_TC: QSPI Transfer complete flag
  - QSPI_FLAG_TE: QSPI Transfer error

flag

Return value:

- None

QSPI Flags

QSPI_FLAG_BUSY	Busy flag: operation is ongoing
QSPI_FLAG_TO	Timeout flag: timeout occurs in memory-mapped mode
QSPI_FLAG_SM	Status match flag: received data matches in autopolling mode
QSPI_FLAG_FT	Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete
QSPI_FLAG_TC	Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted
QSPI_FLAG_TE	Transfer error flag: invalid address is being accessed

QSPI Flash Select

QSPI_FLASH_ID_1	FLASH 1 selected
QSPI_FLASH_ID_2	FLASH 2 selected

QSPI Instruction Mode

QSPI_INSTRUCTION_NONE	No instruction
QSPI_INSTRUCTION_1_LINE	Instruction on a single line
QSPI_INSTRUCTION_2_LINES	Instruction on two lines
QSPI_INSTRUCTION_4_LINES	Instruction on four lines

QSPI Interrupts

QSPI_IT_TO	Interrupt on the timeout flag
QSPI_IT_SM	Interrupt on the status match flag
QSPI_IT_FT	Interrupt on the fifo threshold flag
QSPI_IT_TC	Interrupt on the transfer complete flag
QSPI_IT_TE	Interrupt on the transfer error flag

QSPI Match Mode

QSPI_MATCH_MODE_AND	AND match mode between unmasked bits
QSPI_MATCH_MODE_OR	OR match mode between unmasked bits

QSPI Sample Shifting

QSPI_SAMPLE_SHIFTING_NONE	No clock cycle shift to sample data
QSPI_SAMPLE_SHIFTING_HALFCYCLE	1/2 clock cycle shift to sample data

QSPI Send Instruction Mode

QSPI_SIOO_INST_EVERY_CMD	Send instruction on every transaction
QSPI_SIOO_INST_ONLY_FIRST_CMD	Send instruction only for the first command

QSPI Timeout Activation

QSPI_TIMEOUT_COUNTER_DISABLE	Timeout counter disabled, nCS remains active

QSPI_TIMEOUT_COUNTER_ENABLE	Timeout counter enabled, nCS released when timeout expires

QSPI Timeout definition

HAL_QPSI_TIMEOUT_DEFAULT_VALUE

49 HAL PWR Generic Driver

49.1 PWR Firmware driver registers structures

49.1.1 PWR_PVDTypeDef

Data Fields

- ***uint32_t PVDLevel***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVDTypeDef::PVDLevel***
  PVDLevel: Specifies the PVD detection level. This parameter can be a value of ***PWR_PVD_detection_level***.
- ***uint32_t PWR_PVDTypeDef::Mode***
  Mode: Specifies the operating mode for the selected pins. This parameter can be a value of ***PWR_PVD_Mode***.

49.2 PWR Firmware driver API description

49.2.1 Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_PWR_DeInit()***
- ***HAL_PWR_EnableBkUpAccess()***
- ***HAL_PWR_DisableBkUpAccess()***

49.2.2 Peripheral Control functions

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in PWR_CR2 register).
- PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through __HAL_PVD_EXTI_ENABLE_IT() macro.
- The PVD is stopped in Standby mode.

WakeUp pin configuration

- WakeUp pins are used to wakeup the system from Standby mode or Shutdown mode. The polarity of these pins can be set to configure event detection on high level (rising edge) or low level (falling edge).

Low Power modes configuration

The devices feature 8 low-power modes:

- Low-power Run mode: core and peripherals are running, main regulator off, low power regulator on.
- Sleep mode: Cortex-M4 core stopped, peripherals kept running, main and low power regulators on.
- Low-power Sleep mode: Cortex-M4 core stopped, peripherals kept running, main regulator off, low power regulator on.
- Stop 0 mode: all clocks are stopped except LSI and LSE, main and low power regulators on.
- Stop 1 mode: all clocks are stopped except LSI and LSE, main regulator off, low power regulator on.
- Stop 2 mode: all clocks are stopped except LSI and LSE, main regulator off, low power regulator on, reduced set of waking up IPs compared to Stop 1 mode.
- Standby mode with SRAM2: all clocks are stopped except LSI and LSE, SRAM2 content preserved, main regulator off, low power regulator on.
- Standby mode without SRAM2: all clocks are stopped except LSI and LSE, main and low power regulators off.
- Shutdown mode: all clocks are stopped except LSE, main and low power regulators off.

Low-power run mode

- Entry: (from main run mode)
  - set LPR bit with HAL_PWREx_EnableLowPowerRunMode() API after having decreased the system clock below 2 MHz.
- Exit:
  - clear LPR bit then wait for REGLP bit to be reset with HAL_PWREx_DisableLowPowerRunMode() API. Only then can the system clock frequency be increased above 2 MHz.

Sleep mode / Low-power sleep mode

- Entry: The Sleep mode / Low-power Sleep mode is entered thru HAL_PWR_EnterSLEEPMode() API in specifying whether or not the regulator is forced to low-power mode and if exit is interrupt or event-triggered.
  - PWR_MAINREGULATOR_ON: Sleep mode (regulator in main mode).
  - PWR_LOWPOWERREGULATOR_ON: Low-power sleep (regulator in low power mode). In the latter case, the system clock frequency must have been decreased below 2 MHz beforehand.
  - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
  - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
- WFI Exit:
  - Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) or any wake-up event.
- WFE Exit:
  - Any wake-up event such as an EXTI line configured in event mode.

When exiting the Low-power sleep mode by issuing an interrupt or a wakeup event, the MCU is in Low-power Run mode.

Stop 0, Stop 1 and Stop 2 modes

- Entry: The Stop 0, Stop 1 or Stop 2 modes are entered thru the following APIs:
  - HAL_PWREx_EnterSTOP0Mode() for mode 0 or HAL_PWREx_EnterSTOP1Mode() for mode 1 or for porting reasons HAL_PWR_EnterSTOPMode().

  - HAL_PWREx_EnterSTOP2Mode() for mode 2.
- Regulator setting (applicable to HAL_PWR_EnterSTOPMode() only):
  - PWR_MAINREGULATOR_ON
  - PWR_LOWPOWERREGULATOR_ON
- Exit (interrupt or event-triggered, specified when entering STOP mode):
  - PWR_STOPENTRY_WFI: enter Stop mode with WFI instruction
  - PWR_STOPENTRY_WFE: enter Stop mode with WFE instruction
- WFI Exit:
  - Any EXTI Line (Internal or External) configured in Interrupt mode.
  - Some specific communication peripherals (USART, LPUART, I2C) interrupts when programmed in wakeup mode.
- WFE Exit:
  - Any EXTI Line (Internal or External) configured in Event mode.

When exiting Stop 0 and Stop 1 modes, the MCU is either in Run mode or in Low-power Run mode depending on the LPR bit setting. When exiting Stop 2 mode, the MCU is in Run mode.

Standby mode

The Standby mode offers two options:

- option a) all clocks off except LSI and LSE, RRS bit set (keeps voltage regulator in low power mode). SRAM and registers contents are lost except for the SRAM2 content, the RTC registers, RTC backup registers and Standby circuitry.
- option b) all clocks off except LSI and LSE, RRS bit cleared (voltage regulator then disabled). SRAM and register contents are lost except for the RTC registers, RTC backup registers and Standby circuitry.
  - Entry:
    - The Standby mode is entered thru HAL_PWR_EnterSTANDBYMode() API. SRAM1 and register contents are lost except for registers in the Backup domain and Standby circuitry. SRAM2 content can be preserved if the bit RRS is set in PWR_CR3 register. To enable this feature, the user can resort to HAL_PWREx_EnableSRAM2ContentRetention() API to set RRS bit.
  - Exit:
    - WKUP pin rising edge, RTC alarm or wakeup, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

After waking up from Standby mode, program execution restarts in the same way as after a Reset.

Shutdown mode

In Shutdown mode, voltage regulator is disabled, all clocks are off except LSE, RRS bit is cleared. SRAM and registers contents are lost except for backup domain registers.

- Entry: The Shutdown mode is entered thru HAL_PWREx_EnterSHUTDOWNMode() API.
- Exit:
  - WKUP pin rising edge, RTC alarm or wakeup, tamper event, time-stamp event, external reset in NRST pin.

After waking up from Shutdown mode, program execution restarts in the same way as after a Reset.

Auto-wakeup (AWU) from low-power mode

The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event or a time-stamp event, without depending on an external interrupt (Auto-wakeup mode).

- RTC auto-wakeup (AWU) from the Stop, Standby and Shutdown modes
  - To wake up from the Stop mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL_RTC_SetAlarm_IT() function.
  - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to configure the RTC to detect the tamper or time stamp event using the HAL_RTCEx_SetTimeStamp_IT() or HAL_RTCEx_SetTamper_IT() functions.
  - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL_RTCEx_SetWakeUpTimer_IT() function.

This section contains the following APIs:

- ***HAL_PWR_ConfigPVD()***
- ***HAL_PWR_EnablePVD()***
- ***HAL_PWR_DisablePVD()***
- ***HAL_PWR_EnableWakeUpPin()***
- ***HAL_PWR_DisableWakeUpPin()***
- ***HAL_PWR_EnterSLEEPMode()***
- ***HAL_PWR_EnterSTOPMode()***
- ***HAL_PWR_EnterSTANDBYMode()***
- ***HAL_PWR_EnableSleepOnExit()***
- ***HAL_PWR_DisableSleepOnExit()***
- ***HAL_PWR_EnableSEVOnPend()***
- ***HAL_PWR_DisableSEVOnPend()***
- ***HAL_PWR_PVDCallback()***

49.2.3 Detailed description of functions

HAL_PWR_DeInit

Function name	**void HAL_PWR_DeInit (void)**
Function description	Deinitialize the HAL PWR peripheral registers to their default reset values.
Return values	• **None:**

HAL_PWR_EnableBkUpAccess

Function name	**void HAL_PWR_EnableBkUpAccess (void)**
Function description	Enable access to the backup domain (RTC registers, RTC backup data registers).
Return values	• **None:**
Notes	• After reset, the backup domain is protected against possible unwanted write accesses. • RTCSEL that sets the RTC clock source selection is in the RTC back-up domain. In order to set or modify the RTC clock, the backup domain access must be disabled.

- LSEON bit that switches on and off the LSE crystal belongs as well to the back-up domain.

HAL_PWR_DisableBkUpAccess

Function name	**void HAL_PWR_DisableBkUpAccess (void)**
Function description	Disable access to the backup domain (RTC registers, RTC backup data registers).
Return values	• **None:**

HAL_PWR_ConfigPVD

Function name	**HAL_StatusTypeDef HAL_PWR_ConfigPVD (PWR_PVDTypeDef * sConfigPVD)**
Function description	Configure the voltage threshold detected by the Power Voltage Detector (PVD).
Parameters	• **sConfigPVD:** pointer to a PWR_PVDTypeDef structure that contains the PVD configuration information.
Return values	• **None:**
Notes	• Refer to the electrical characteristics of your device datasheet for more details about the voltage thresholds corresponding to each detection level.

HAL_PWR_EnablePVD

Function name	**void HAL_PWR_EnablePVD (void)**
Function description	Enable the Power Voltage Detector (PVD).
Return values	• **None:**

HAL_PWR_DisablePVD

Function name	**void HAL_PWR_DisablePVD (void)**
Function description	Disable the Power Voltage Detector (PVD).
Return values	• **None:**

HAL_PWR_EnableWakeUpPin

Function name	**void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinPolarity)**
Function description	Enable the WakeUp PINx functionality.
Parameters	• **WakeUpPinPolarity:** Specifies which Wake-Up pin to enable. This parameter can be one of the following legacy values which set the default polarity i.e. detection on high level (rising edge): or one of the following value where the user can explicitly specify the enabled pin and the chosen polarity: – PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4,

PWR_WAKEUP_PIN5
  - PWR_WAKEUP_PIN1_HIGH or PWR_WAKEUP_PIN1_LOW
  - PWR_WAKEUP_PIN2_HIGH or PWR_WAKEUP_PIN2_LOW
  - PWR_WAKEUP_PIN3_HIGH or PWR_WAKEUP_PIN3_LOW
  - PWR_WAKEUP_PIN4_HIGH or PWR_WAKEUP_PIN4_LOW
  - PWR_WAKEUP_PIN5_HIGH or PWR_WAKEUP_PIN5_LOW

Return values	• **None:**
Notes	• PWR_WAKEUP_PINx and PWR_WAKEUP_PINx_HIGH are equivalent.

HAL_PWR_DisableWakeUpPin

Function name	**void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)**
Function description	Disable the WakeUp PINx functionality.
Parameters	• **WakeUpPinx:** Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values: – PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN5
Return values	• **None:**

HAL_PWR_EnterSLEEPMode

Function name	**void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)**
Function description	Enter Sleep or Low-power Sleep mode.
Parameters	• **Regulator:** Specifies the regulator state in Sleep/Low-power Sleep mode. This parameter can be one of the following values: – PWR_MAINREGULATOR_ON Sleep mode (regulator in main mode) – PWR_LOWPOWERREGULATOR_ON Low-power Sleep mode (regulator in low-power mode) • **SLEEPEntry:** Specifies if Sleep mode is entered with WFI or WFE instruction. This parameter can be one of the following values: – PWR_SLEEPENTRY_WFI enter Sleep or Low-power Sleep mode with WFI instruction – PWR_SLEEPENTRY_WFE enter Sleep or Low-power Sleep mode with WFE instruction
Return values	• **None:**
Notes	• In Sleep/Low-power Sleep mode, all I/O pins keep the same state as in Run mode. • Low-power Sleep mode is entered from Low-power Run

mode. Therefore, if not yet in Low-power Run mode before calling HAL_PWR_EnterSLEEPMode() with Regulator set to PWR_LOWPOWERREGULATOR_ON, the user can optionally configure the Flash in power-down monde in setting the SLEEP_PD bit in FLASH_ACR register. Additionally, the clock frequency must be reduced below 2 MHz. Setting SLEEP_PD in FLASH_ACR then appropriately reducing the clock frequency must be done before calling HAL_PWR_EnterSLEEPMode() API.
- When exiting Low-power Sleep mode, the MCU is in Low-power Run mode. To move in Run mode, the user must resort to HAL_PWREx_DisableLowPowerRunMode() API.
- When WFI entry is used, tick interrupt have to be disabled if not desired as the interrupt wake up source.

HAL_PWR_EnterSTOPMode

Function name
: **void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)**

Function description
: Enter Stop mode.

Parameters
- **Regulator:** Specifies the regulator state in Stop mode. This parameter can be one of the following values:
  - PWR_MAINREGULATOR_ON Stop 0 mode (main regulator ON)
  - PWR_LOWPOWERREGULATOR_ON Stop 1 mode (low power regulator ON)
- **STOPEntry:** Specifies Stop 0 or Stop 1 mode is entered with WFI or WFE instruction. This parameter can be one of the following values:
  - PWR_STOPENTRY_WFI Enter Stop 0 or Stop 1 mode with WFI instruction.
  - PWR_STOPENTRY_WFE Enter Stop 0 or Stop 1 mode with WFE instruction.

Return values
- **None:**

Notes
- This API is named HAL_PWR_EnterSTOPMode to ensure compatibility with legacy code running on devices where only "Stop mode" is mentioned with main or low power regulator ON.
- In Stop mode, all I/O pins keep the same state as in Run mode.
- All clocks in the VCORE domain are stopped; the PLL, the MSI, the HSI and the HSE oscillators are disabled. Some peripherals with the wakeup capability (I2Cx, USARTx and LPUART) can switch on the HSI to receive a frame, and switch off the HSI after receiving the frame if it is not a wakeup frame. In this case, the HSI clock is propagated only to the peripheral requesting it. SRAM1, SRAM2 and register contents are preserved. The BOR is available. The voltage regulator can be configured either in normal (Stop 0) or low-power mode (Stop 1).
- When exiting Stop 0 or Stop 1 mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system

clock if STOPWUCK bit in RCC_CFGR register is set; the MSI oscillator is selected if STOPWUCK is cleared.
- When the voltage regulator operates in low power mode (Stop 1), an additional startup delay is incurred when waking up. By keeping the internal regulator ON during Stop mode (Stop 0), the consumption is higher although the startup time is reduced.

HAL_PWR_EnterSTANDBYMode

Function name	**void HAL_PWR_EnterSTANDBYMode (void)**
Function description	Enter Standby mode.
Return values	• **None:**
Notes	• In Standby mode, the PLL, the HSI, the MSI and the HSE oscillators are switched off. The voltage regulator is disabled, except when SRAM2 content is preserved in which case the regulator is in low-power mode. SRAM1 and register contents are lost except for registers in the Backup domain and Standby circuitry. SRAM2 content can be preserved if the bit RRS is set in PWR_CR3 register. To enable this feature, the user can resort to HAL_PWREx_EnableSRAM2ContentRetention() API to set RRS bit. The BOR is available. • The I/Os can be configured either with a pull-up or pull-down or can be kept in analog state. HAL_PWREx_EnableGPIOPullUp() and HAL_PWREx_EnableGPIOPullDown() respectively enable Pull Up and Pull Down state, HAL_PWREx_DisableGPIOPullUp() and HAL_PWREx_DisableGPIOPullDown() disable the same. These states are effective in Standby mode only if APC bit is set through HAL_PWREx_EnablePullUpPullDownConfig() API.

HAL_PWR_EnableSleepOnExit

Function name	**void HAL_PWR_EnableSleepOnExit (void)**
Function description	Indicate Sleep-On-Exit when returning from Handler mode to Thread mode.
Return values	• **None:**
Notes	• Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.

HAL_PWR_DisableSleepOnExit

Function name	**void HAL_PWR_DisableSleepOnExit (void)**
Function description	Disable Sleep-On-Exit feature when returning from Handler mode to Thread mode.

Return values	• **None:**
Notes	• Clear SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name	**void HAL_PWR_EnableSEVOnPend (void)**
Function description	Enable CORTEX M4 SEVONPEND bit.
Return values	• **None:**
Notes	• Set SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

HAL_PWR_DisableSEVOnPend

Function name	**void HAL_PWR_DisableSEVOnPend (void)**
Function description	Disable CORTEX M4 SEVONPEND bit.
Return values	• **None:**
Notes	• Clear SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

HAL_PWR_PVDCallback

Function name	**void HAL_PWR_PVDCallback (void)**
Function description	PWR PVD interrupt callback.
Return values	• **None:**

49.3 PWR Firmware driver defines

49.3.1 PWR

PWR Exported Macros

__HAL_PWR_GET_FLAG

Description:

- Check whether or not a specific PWR flag is set.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - PWR_FLAG_WUF1 Wake Up Flag 1. Indicates that a wakeup event was received from the WKUP pin 1.
  - PWR_FLAG_WUF2 Wake Up Flag 2. Indicates that a wakeup

event was received from the WKUP pin 2.
- PWR_FLAG_WUF3 Wake Up Flag 3. Indicates that a wakeup event was received from the WKUP pin 3.
- PWR_FLAG_WUF4 Wake Up Flag 4. Indicates that a wakeup event was received from the WKUP pin 4.
- PWR_FLAG_WUF5 Wake Up Flag 5. Indicates that a wakeup event was received from the WKUP pin 5.
- PWR_FLAG_SB StandBy Flag. Indicates that the system entered StandBy mode.
- PWR_FLAG_WUFI Wake-Up Flag Internal. Set when a wakeup is detected on the internal wakeup line.
- PWR_FLAG_REGLPS Low Power Regulator Started. Indicates whether or not the low-power regulator is ready.
- PWR_FLAG_REGLPF Low Power Regulator Flag. Indicates whether the regulator is ready in main mode or is in low-power mode.
- PWR_FLAG_VOSF Voltage Scaling Flag. Indicates whether the regulator is ready in the selected voltage range or is still changing to the required voltage level.
- PWR_FLAG_PVDO Power Voltage Detector Output. Indicates whether VDD voltage is below or above the selected PVD threshold.
- PWR_FLAG_PVMO1 Peripheral Voltage Monitoring Output 1. Indicates whether VDDUSB voltage is is below or above PVM1 threshold (applicable when USB feature is supported).
- PWR_FLAG_PVMO3 Peripheral Voltage Monitoring Output 3. Indicates whether VDDA voltage is is below or above PVM3 threshold.
- PWR_FLAG_PVMO4 Peripheral Voltage Monitoring

Output 4. Indicates whether VDDA voltage is is below or above PVM4 threshold.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_PWR_CLEAR_FLAG

Description:

- Clear a specific PWR flag.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - PWR_FLAG_WUF1 Wake Up Flag 1. Indicates that a wakeup event was received from the WKUP pin 1.
  - PWR_FLAG_WUF2 Wake Up Flag 2. Indicates that a wakeup event was received from the WKUP pin 2.
  - PWR_FLAG_WUF3 Wake Up Flag 3. Indicates that a wakeup event was received from the WKUP pin 3.
  - PWR_FLAG_WUF4 Wake Up Flag 4. Indicates that a wakeup event was received from the WKUP pin 4.
  - PWR_FLAG_WUF5 Wake Up Flag 5. Indicates that a wakeup event was received from the WKUP pin 5.
  - PWR_FLAG_WU Encompasses all five Wake Up Flags.
  - PWR_FLAG_SB Standby Flag. Indicates that the system entered Standby mode.

Return value:

- None

__HAL_PWR_PVD_EXTI_ENABLE_IT

Description:

- Enable the PVD Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVD_EXTI_DISABLE_IT

Description:

- Disable the PVD Extended Interrupt

Line.

Return value:

- None

__HAL_PWR_PVD_EXTI_ENABLE_EVENT

Description:

- Enable the PVD Event Line.

Return value:

- None

__HAL_PWR_PVD_EXTI_DISABLE_EVENT

Description:

- Disable the PVD Event Line.

Return value:

- None

__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVD Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVD Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVD Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVD Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVD Extended Interrupt

Rising & Falling Trigger.

	Return value: • None
__HAL_PWR_PVD_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on selected EXTI line. **Return value:** • None
__HAL_PWR_PVD_EXTI_GET_FLAG	**Description:** • Check whether or not the PVD EXTI interrupt flag is set. **Return value:** • EXTI: PVD Line Status.
__HAL_PWR_PVD_EXTI_CLEAR_FLAG	**Description:** • Clear the PVD EXTI interrupt flag. **Return value:** • None

Programmable Voltage Detection levels

PWR_PVDLEVEL_0	PVD threshold around 2.0 V
PWR_PVDLEVEL_1	PVD threshold around 2.2 V
PWR_PVDLEVEL_2	PVD threshold around 2.4 V
PWR_PVDLEVEL_3	PVD threshold around 2.5 V
PWR_PVDLEVEL_4	PVD threshold around 2.6 V
PWR_PVDLEVEL_5	PVD threshold around 2.8 V
PWR_PVDLEVEL_6	PVD threshold around 2.9 V
PWR_PVDLEVEL_7	External input analog voltage (compared internally to VREFINT)

PWR PVD event line

PWR_EVENT_LINE_PVD	Event line 16 Connected to the PVD Event Line

PWR PVD external interrupt line

PWR_EXTI_LINE_PVD	External interrupt line 16 Connected to the PVD EXTI Line

PWR PVD interrupt and event mode

PWR_PVD_MODE_NORMAL	Basic mode is used
PWR_PVD_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
PWR_PVD_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
PWR_PVD_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING	Event Mode with Rising edge trigger detection
PWR_PVD_MODE_EVENT_FALLING	Event Mode with Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING_FALLING	Event Mode with Rising/Falling edge trigger detection

PWR PVD Mode Mask

PVD_MODE_IT	Mask for interruption yielded by PVD threshold crossing
PVD_MODE_EVT	Mask for event yielded by PVD threshold crossing
PVD_RISING_EDGE	Mask for rising edge set as PVD trigger
PVD_FALLING_EDGE	Mask for falling edge set as PVD trigger

PWR regulator mode

PWR_MAINREGULATOR_ON	Regulator in main mode
PWR_LOWPOWERREGULATOR_ON	Regulator in low-power mode

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI	Wait For Interruption instruction to enter Sleep mode
PWR_SLEEPENTRY_WFE	Wait For Event instruction to enter Sleep mode

PWR STOP mode entry

PWR_STOPENTRY_WFI	Wait For Interruption instruction to enter Stop mode
PWR_STOPENTRY_WFE	Wait For Event instruction to enter Stop mode

50 HAL PWR Extension Driver

50.1 PWREx Firmware driver registers structures

50.1.1 PWR_PVMTypeDef

Data Fields

- ***uint32_t PVMType***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVMTypeDef::PVMType***
  PVMType: Specifies which voltage is monitored and against which threshold. This parameter can be a value of ***PWREx_PVM_Type***. **PWR_PVM_1** Peripheral Voltage Monitoring 1 enable: VDDUSB versus 1.2 V (applicable when USB feature is supported). **PWR_PVM_3** Peripheral Voltage Monitoring 3 enable: VDDA versus 1.62 V. **PWR_PVM_4** Peripheral Voltage Monitoring 4 enable: VDDA versus 2.2 V.
- ***uint32_t PWR_PVMTypeDef::Mode***
  Mode: Specifies the operating mode for the selected pins. This parameter can be a value of ***PWREx_PVM_Mode***.

50.2 PWREx Firmware driver API description

50.2.1 Extended Peripheral Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_PWREx_GetVoltageRange()***
- ***HAL_PWREx_ControlVoltageScaling()***
- ***HAL_PWREx_EnableBatteryCharging()***
- ***HAL_PWREx_DisableBatteryCharging()***
- ***HAL_PWREx_EnableVddUSB()***
- ***HAL_PWREx_DisableVddUSB()***
- ***HAL_PWREx_EnableVddIO2()***
- ***HAL_PWREx_DisableVddIO2()***
- ***HAL_PWREx_EnableInternalWakeUpLine()***
- ***HAL_PWREx_DisableInternalWakeUpLine()***
- ***HAL_PWREx_EnableGPIOPullUp()***
- ***HAL_PWREx_DisableGPIOPullUp()***
- ***HAL_PWREx_EnableGPIOPullDown()***
- ***HAL_PWREx_DisableGPIOPullDown()***
- ***HAL_PWREx_EnablePullUpPullDownConfig()***
- ***HAL_PWREx_DisablePullUpPullDownConfig()***
- ***HAL_PWREx_EnableSRAM2ContentRetention()***
- ***HAL_PWREx_DisableSRAM2ContentRetention()***
- ***HAL_PWREx_EnableSRAM3ContentRetention()***
- ***HAL_PWREx_DisableSRAM3ContentRetention()***
- ***HAL_PWREx_EnableDSIPinsPDActivation()***
- ***HAL_PWREx_DisableDSIPinsPDActivation()***

- *HAL_PWREx_EnablePVM1()*
- *HAL_PWREx_DisablePVM1()*
- *HAL_PWREx_EnablePVM2()*
- *HAL_PWREx_DisablePVM2()*
- *HAL_PWREx_EnablePVM3()*
- *HAL_PWREx_DisablePVM3()*
- *HAL_PWREx_EnablePVM4()*
- *HAL_PWREx_DisablePVM4()*
- *HAL_PWREx_ConfigPVM()*
- *HAL_PWREx_EnableLowPowerRunMode()*
- *HAL_PWREx_DisableLowPowerRunMode()*
- *HAL_PWREx_EnterSTOP0Mode()*
- *HAL_PWREx_EnterSTOP1Mode()*
- *HAL_PWREx_EnterSTOP2Mode()*
- *HAL_PWREx_EnterSHUTDOWNMode()*
- *HAL_PWREx_PVD_PVM_IRQHandler()*
- *HAL_PWREx_PVM1Callback()*
- *HAL_PWREx_PVM2Callback()*
- *HAL_PWREx_PVM3Callback()*
- *HAL_PWREx_PVM4Callback()*

50.2.2 Detailed description of functions

HAL_PWREx_GetVoltageRange

Function name	**uint32_t HAL_PWREx_GetVoltageRange (void)**
Function description	Return Voltage Scaling Range.
Return values	• **VOS:** bit field (PWR_REGULATOR_VOLTAGE_RANGE1 or PWR_REGULATOR_VOLTAGE_RANGE2 or PWR_REGULATOR_VOLTAGE_SCALE1_BOOST when applicable)

HAL_PWREx_ControlVoltageScaling

Function name	**HAL_StatusTypeDef HAL_PWREx_ControlVoltageScaling (uint32_t VoltageScaling)**
Function description	Configure the main internal regulator output voltage.
Parameters	• **VoltageScaling:** specifies the regulator output voltage to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values: – PWR_REGULATOR_VOLTAGE_SCALE1_BOOST when available, Regulator voltage output range 1 boost mode, typical output voltage at 1.2 V, system frequency up to 120 MHz. – PWR_REGULATOR_VOLTAGE_SCALE1 Regulator voltage output range 1 mode, typical output voltage at 1.2 V, system frequency up to 80 MHz. – PWR_REGULATOR_VOLTAGE_SCALE2 Regulator voltage output range 2 mode, typical output voltage at 1.0 V, system frequency up to 26 MHz.

Return values	• **HAL:** Status
Notes	• When moving from Range 1 to Range 2, the system frequency must be decreased to a value below 26 MHz before calling HAL_PWREx_ControlVoltageScaling() API. When moving from Range 2 to Range 1, the system frequency can be increased to a value up to 80 MHz after calling HAL_PWREx_ControlVoltageScaling() API. For some devices, the system frequency can be increased up to 120 MHz. • When moving from Range 2 to Range 1, the API waits for VOSF flag to be cleared before returning the status. If the flag is not cleared within 50 microseconds, HAL_TIMEOUT status is reported.

HAL_PWREx_EnableBatteryCharging

Function name	**void HAL_PWREx_EnableBatteryCharging (uint32_t ResistorSelection)**
Function description	Enable battery charging.
Parameters	• **ResistorSelection:** specifies the resistor impedance. This parameter can be one of the following values: – PWR_BATTERY_CHARGING_RESISTOR_5 5 kOhms resistor – PWR_BATTERY_CHARGING_RESISTOR_1_5 1.5 kOhms resistor
Return values	• **None:**

HAL_PWREx_DisableBatteryCharging

Function name	**void HAL_PWREx_DisableBatteryCharging (void)**
Function description	Disable battery charging.
Return values	• **None:**

HAL_PWREx_EnableVddUSB

Function name	**void HAL_PWREx_EnableVddUSB (void)**
Function description	Enable VDDUSB supply.
Return values	• **None:**
Notes	• Remove VDDUSB electrical and logical isolation, once VDDUSB supply is present.

HAL_PWREx_DisableVddUSB

Function name	**void HAL_PWREx_DisableVddUSB (void)**
Function description	Disable VDDUSB supply.
Return values	• **None:**

HAL_PWREx_EnableVddIO2

Function name	**void HAL_PWREx_EnableVddIO2 (void)**
Function description	Enable VDDIO2 supply.
Return values	• **None:**
Notes	• Remove VDDIO2 electrical and logical isolation, once VDDIO2 supply is present.

HAL_PWREx_DisableVddIO2

Function name	**void HAL_PWREx_DisableVddIO2 (void)**
Function description	Disable VDDIO2 supply.
Return values	• **None:**

HAL_PWREx_EnableInternalWakeUpLine

Function name	**void HAL_PWREx_EnableInternalWakeUpLine (void)**
Function description	Enable Internal Wake-up Line.
Return values	• **None:**

HAL_PWREx_DisableInternalWakeUpLine

Function name	**void HAL_PWREx_DisableInternalWakeUpLine (void)**
Function description	Disable Internal Wake-up Line.
Return values	• **None:**

HAL_PWREx_EnableGPIOPullUp

Function name	**HAL_StatusTypeDef HAL_PWREx_EnableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)**
Function description	Enable GPIO pull-up state in Standby and Shutdown modes.
Parameters	• **GPIO:** Specify the IO port. This parameter can be PWR_GPIO_A, ..., PWR_GPIO_H (or PWR_GPIO_I depending on the devices) to select the GPIO peripheral. • **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for the port where less I/O pins are available) or the logical OR of several of them to set several bits for a given port in a single API call.
Return values	• **HAL:** Status
Notes	• Set the relevant PUy bits of PWR_PUCRx register to configure the I/O in pull-up state in Standby and Shutdown modes. • This state is effective in Standby and Shutdown modes only if APC bit is set through HAL_PWREx_EnablePullUpPullDownConfig() API. • The configuration is lost when exiting the Shutdown mode due to the power-on reset, maintained when exiting the

Standby mode.
- To avoid any conflict at Standby and Shutdown modes exits, the corresponding PDy bit of PWR_PDCRx register is cleared unless it is reserved.
- Even if a PUy bit to set is reserved, the other PUy bits entered as input parameter at the same time are set.

HAL_PWREx_DisableGPIOPullUp

Function name	**HAL_StatusTypeDef HAL_PWREx_DisableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)**
Function description	Disable GPIO pull-up state in Standby mode and Shutdown modes.
Parameters	• **GPIO:** Specifies the IO port. This parameter can be PWR_GPIO_A, ..., PWR_GPIO_H (or PWR_GPIO_I depending on the devices) to select the GPIO peripheral. • **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for the port where less I/O pins are available) or the logical OR of several of them to reset several bits for a given port in a single API call.
Return values	• **HAL:** Status
Notes	• Reset the relevant PUy bits of PWR_PUCRx register used to configure the I/O in pull-up state in Standby and Shutdown modes. • Even if a PUy bit to reset is reserved, the other PUy bits entered as input parameter at the same time are reset.

HAL_PWREx_EnableGPIOPullDown

Function name	**HAL_StatusTypeDef HAL_PWREx_EnableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)**
Function description	Enable GPIO pull-down state in Standby and Shutdown modes.
Parameters	• **GPIO:** Specify the IO port. This parameter can be PWR_GPIO_A..PWR_GPIO_H (or PWR_GPIO_I depending on the devices) to select the GPIO peripheral. • **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for the port where less I/O pins are available) or the logical OR of several of them to set several bits for a given port in a single API call.
Return values	• **HAL:** Status
Notes	• Set the relevant PDy bits of PWR_PDCRx register to configure the I/O in pull-down state in Standby and Shutdown modes. • This state is effective in Standby and Shutdown modes only if APC bit is set through HAL_PWREx_EnablePullUpPullDownConfig() API. • The configuration is lost when exiting the Shutdown mode due to the power-on reset, maintained when exiting the

Standby mode.

- To avoid any conflict at Standby and Shutdown modes exits, the corresponding PUy bit of PWR_PUCRx register is cleared unless it is reserved.
- Even if a PDy bit to set is reserved, the other PDy bits entered as input parameter at the same time are set.

HAL_PWREx_DisableGPIOPullDown

Function name	**HAL_StatusTypeDef HAL_PWREx_DisableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)**
Function description	Disable GPIO pull-down state in Standby and Shutdown modes.
Parameters	• **GPIO:** Specifies the IO port. This parameter can be PWR_GPIO_A..PWR_GPIO_H (or PWR_GPIO_I depending on the devices) to select the GPIO peripheral. • **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for the port where less I/O pins are available) or the logical OR of several of them to reset several bits for a given port in a single API call.
Return values	• **HAL:** Status
Notes	• Reset the relevant PDy bits of PWR_PDCRx register used to configure the I/O in pull-down state in Standby and Shutdown modes. • Even if a PDy bit to reset is reserved, the other PDy bits entered as input parameter at the same time are reset.

HAL_PWREx_EnablePullUpPullDownConfig

Function name	**void HAL_PWREx_EnablePullUpPullDownConfig (void)**
Function description	Enable pull-up and pull-down configuration.
Return values	• **None:**
Notes	• When APC bit is set, the I/O pull-up and pull-down configurations defined in PWR_PUCRx and PWR_PDCRx registers are applied in Standby and Shutdown modes. • Pull-up set by PUy bit of PWR_PUCRx register is not activated if the corresponding PDy bit of PWR_PDCRx register is also set (pull-down configuration priority is higher). HAL_PWREx_EnableGPIOPullUp() and HAL_PWREx_EnableGPIOPullDown() APIs ensure there is no conflict when setting PUy or PDy bit.

HAL_PWREx_DisablePullUpPullDownConfig

Function name	**void HAL_PWREx_DisablePullUpPullDownConfig (void)**
Function description	Disable pull-up and pull-down configuration.
Return values	• **None:**
Notes	• When APC bit is cleared, the I/O pull-up and pull-down configurations defined in PWR_PUCRx and PWR_PDCRx

registers are not applied in Standby and Shutdown modes.

HAL_PWREx_EnableSRAM2ContentRetention

Function name	**void HAL_PWREx_EnableSRAM2ContentRetention (void)**
Function description	Enable SRAM2 content retention in Standby mode.
Return values	• **None:**
Notes	• When RRS bit is set, SRAM2 is powered by the low-power regulator in Standby mode and its content is kept.

HAL_PWREx_DisableSRAM2ContentRetention

Function name	**void HAL_PWREx_DisableSRAM2ContentRetention (void)**
Function description	Disable SRAM2 content retention in Standby mode.
Return values	• **None:**
Notes	• When RRS bit is reset, SRAM2 is powered off in Standby mode and its content is lost.

HAL_PWREx_EnableSRAM3ContentRetention

Function name	**void HAL_PWREx_EnableSRAM3ContentRetention (void)**
Function description	Enable SRAM3 content retention in Stop 2 mode.
Return values	• **None:**
Notes	• When RRSTP bit is set, SRAM3 is powered by the low-power regulator in Stop 2 mode and its content is kept.

HAL_PWREx_DisableSRAM3ContentRetention

Function name	**void HAL_PWREx_DisableSRAM3ContentRetention (void)**
Function description	Disable SRAM3 content retention in Stop 2 mode.
Return values	• **None:**
Notes	• When RRSTP bit is reset, SRAM3 is powered off in Stop 2 mode and its content is lost.

HAL_PWREx_EnableDSIPinsPDActivation

Function name	**void HAL_PWREx_EnableDSIPinsPDActivation (void)**
Function description	Enable pull-down activation on DSI pins.
Return values	• **None:**

HAL_PWREx_DisableDSIPinsPDActivation

Function name	**void HAL_PWREx_DisableDSIPinsPDActivation (void)**
Function description	Disable pull-down activation on DSI pins.
Return values	• **None:**

HAL_PWREx_EnablePVM1

Function name	**void HAL_PWREx_EnablePVM1 (void)**
Function description	Enable the Power Voltage Monitoring 1: VDDUSB versus 1.2V.
Return values	• **None:**

HAL_PWREx_DisablePVM1

Function name	**void HAL_PWREx_DisablePVM1 (void)**
Function description	Disable the Power Voltage Monitoring 1: VDDUSB versus 1.2V.
Return values	• **None:**

HAL_PWREx_EnablePVM2

Function name	**void HAL_PWREx_EnablePVM2 (void)**
Function description	Enable the Power Voltage Monitoring 2: VDDIO2 versus 0.9V.
Return values	• **None:**

HAL_PWREx_DisablePVM2

Function name	**void HAL_PWREx_DisablePVM2 (void)**
Function description	Disable the Power Voltage Monitoring 2: VDDIO2 versus 0.9V.
Return values	• **None:**

HAL_PWREx_EnablePVM3

Function name	**void HAL_PWREx_EnablePVM3 (void)**
Function description	Enable the Power Voltage Monitoring 3: VDDA versus 1.62V.
Return values	• **None:**

HAL_PWREx_DisablePVM3

Function name	**void HAL_PWREx_DisablePVM3 (void)**
Function description	Disable the Power Voltage Monitoring 3: VDDA versus 1.62V.
Return values	• **None:**

HAL_PWREx_EnablePVM4

Function name	**void HAL_PWREx_EnablePVM4 (void)**
Function description	Enable the Power Voltage Monitoring 4: VDDA versus 2.2V.
Return values	• **None:**

HAL_PWREx_DisablePVM4

Function name	**void HAL_PWREx_DisablePVM4 (void)**
Function description	Disable the Power Voltage Monitoring 4: VDDA versus 2.2V.

Return values	• **None:**

HAL_PWREx_ConfigPVM

Function name	**HAL_StatusTypeDef HAL_PWREx_ConfigPVM (PWR_PVMTypeDef * sConfigPVM)**
Function description	Configure the Peripheral Voltage Monitoring (PVM).
Parameters	• **sConfigPVM:** pointer to a PWR_PVMTypeDef structure that contains the PVM configuration information.
Return values	• **HAL:** status
Notes	• The API configures a single PVM according to the information contained in the input structure. To configure several PVMs, the API must be singly called for each PVM used. • Refer to the electrical characteristics of your device datasheet for more details about the voltage thresholds corresponding to each detection level and to each monitored supply.

HAL_PWREx_EnableLowPowerRunMode

Function name	**void HAL_PWREx_EnableLowPowerRunMode (void)**
Function description	Enter Low-power Run mode.
Return values	• **None:**
Notes	• In Low-power Run mode, all I/O pins keep the same state as in Run mode. • When Regulator is set to PWR_LOWPOWERREGULATOR_ON, the user can optionally configure the Flash in power-down monde in setting the RUN_PD bit in FLASH_ACR register. Additionally, the clock frequency must be reduced below 2 MHz. Setting RUN_PD in FLASH_ACR then appropriately reducing the clock frequency must be done before calling HAL_PWREx_EnableLowPowerRunMode() API.

HAL_PWREx_DisableLowPowerRunMode

Function name	**HAL_StatusTypeDef HAL_PWREx_DisableLowPowerRunMode (void)**
Function description	Exit Low-power Run mode.
Return values	• **HAL:** Status
Notes	• Before HAL_PWREx_DisableLowPowerRunMode() completion, the function checks that REGLPF has been properly reset (otherwise, HAL_PWREx_DisableLowPowerRunMode returns HAL_TIMEOUT status). The system clock frequency can then be increased above 2 MHz.

HAL_PWREx_EnterSTOP0Mode

Function name	**void HAL_PWREx_EnterSTOP0Mode (uint8_t STOPEntry)**
Function description	Enter Stop 0 mode.
Parameters	• **STOPEntry:** specifies if Stop mode in entered with WFI or WFE instruction. This parameter can be one of the following values: – PWR_STOPENTRY_WFI Enter Stop mode with WFI instruction – PWR_STOPENTRY_WFE Enter Stop mode with WFE instruction
Return values	• **None:**
Notes	• In Stop 0 mode, main and low voltage regulators are ON. • In Stop 0 mode, all I/O pins keep the same state as in Run mode. • All clocks in the VCORE domain are stopped; the PLL, the MSI, the HSI and the HSE oscillators are disabled. Some peripherals with the wakeup capability (I2Cx, USARTx and LPUART) can switch on the HSI to receive a frame, and switch off the HSI after receiving the frame if it is not a wakeup frame. In this case, the HSI clock is propagated only to the peripheral requesting it. SRAM1, SRAM2 and register contents are preserved. The BOR is available. • When exiting Stop 0 mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock if STOPWUCK bit in RCC_CFGR register is set; the MSI oscillator is selected if STOPWUCK is cleared. • By keeping the internal regulator ON during Stop 0 mode, the consumption is higher although the startup time is reduced.

HAL_PWREx_EnterSTOP1Mode

Function name	**void HAL_PWREx_EnterSTOP1Mode (uint8_t STOPEntry)**
Function description	Enter Stop 1 mode.
Parameters	• **STOPEntry:** specifies if Stop mode in entered with WFI or WFE instruction. This parameter can be one of the following values: – PWR_STOPENTRY_WFI Enter Stop mode with WFI instruction – PWR_STOPENTRY_WFE Enter Stop mode with WFE instruction
Return values	• **None:**
Notes	• In Stop 1 mode, only low power voltage regulator is ON. • In Stop 1 mode, all I/O pins keep the same state as in Run mode. • All clocks in the VCORE domain are stopped; the PLL, the MSI, the HSI and the HSE oscillators are disabled. Some peripherals with the wakeup capability (I2Cx, USARTx and LPUART) can switch on the HSI to receive a frame, and switch off the HSI after receiving the frame if it is not a wakeup frame. In this case, the HSI clock is propagated only

to the peripheral requesting it. SRAM1, SRAM2 and register contents are preserved. The BOR is available.
- When exiting Stop 1 mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock if STOPWUCK bit in RCC_CFGR register is set; the MSI oscillator is selected if STOPWUCK is cleared.
- Due to low power mode, an additional startup delay is incurred when waking up from Stop 1 mode.

HAL_PWREx_EnterSTOP2Mode

Function name: **void HAL_PWREx_EnterSTOP2Mode (uint8_t STOPEntry)**

Function description: Enter Stop 2 mode.

Parameters:
- **STOPEntry:** specifies if Stop mode in entered with WFI or WFE instruction. This parameter can be one of the following values:
  - PWR_STOPENTRY_WFI Enter Stop mode with WFI instruction
  - PWR_STOPENTRY_WFE Enter Stop mode with WFE instruction

Return values:
- **None:**

Notes:
- In Stop 2 mode, only low power voltage regulator is ON.
- In Stop 2 mode, all I/O pins keep the same state as in Run mode.
- All clocks in the VCORE domain are stopped, the PLL, the MSI, the HSI and the HSE oscillators are disabled. Some peripherals with wakeup capability (LCD, LPTIM1, I2C3 and LPUART) can switch on the HSI to receive a frame, and switch off the HSI after receiving the frame if it is not a wakeup frame. In this case the HSI clock is propagated only to the peripheral requesting it. SRAM1, SRAM2 and register contents are preserved. The BOR is available. The voltage regulator is set in low-power mode but LPR bit must be cleared to enter stop 2 mode. Otherwise, Stop 1 mode is entered.
- When exiting Stop 2 mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock if STOPWUCK bit in RCC_CFGR register is set; the MSI oscillator is selected if STOPWUCK is cleared.

HAL_PWREx_EnterSHUTDOWNMode

Function name: **void HAL_PWREx_EnterSHUTDOWNMode (void)**

Function description: Enter Shutdown mode.

Return values:
- **None:**

Notes:
- In Shutdown mode, the PLL, the HSI, the MSI, the LSI and the HSE oscillators are switched off. The voltage regulator is disabled and Vcore domain is powered off. SRAM1, SRAM2 and registers contents are lost except for registers in the Backup domain. The BOR is not available.
- The I/Os can be configured either with a pull-up or pull-down

or can be kept in analog state.

HAL_PWREx_PVD_PVM_IRQHandler

Function name	**void HAL_PWREx_PVD_PVM_IRQHandler (void)**
Function description	This function handles the PWR PVD/PVMx interrupt request.
Return values	• **None:**
Notes	• This API should be called under the PVD_PVM_IRQHandler().

HAL_PWREx_PVM1Callback

Function name	**void HAL_PWREx_PVM1Callback (void)**
Function description	PWR PVM1 interrupt callback.
Return values	• **None:**

HAL_PWREx_PVM2Callback

Function name	**void HAL_PWREx_PVM2Callback (void)**
Function description	PWR PVM2 interrupt callback.
Return values	• **None:**

HAL_PWREx_PVM3Callback

Function name	**void HAL_PWREx_PVM3Callback (void)**
Function description	PWR PVM3 interrupt callback.
Return values	• **None:**

HAL_PWREx_PVM4Callback

Function name	**void HAL_PWREx_PVM4Callback (void)**
Function description	PWR PVM4 interrupt callback.
Return values	• **None:**

50.3 PWREx Firmware driver defines

50.3.1 PWREx

PWR Extended Exported Macros

__HAL_PWR_PVM1_EXTI_ENABLE_IT	**Description:** • Enable the PVM1 Extended Interrupt Line. **Return value:** • None
__HAL_PWR_PVM1_EXTI_DISABLE_IT	**Description:**

- Disable the PVM1 Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVM1_EXTI_ENABLE_EVENT

Description:

- Enable the PVM1 Event Line.

Return value:

- None

__HAL_PWR_PVM1_EXTI_DISABLE_EVENT

Description:

- Disable the PVM1 Event Line.

Return value:

- None

__HAL_PWR_PVM1_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVM1 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVM1 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVM1 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVM1 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- PVM1 EXTI line configuration: set rising & falling edge trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_DISABLE_RISING

Description:

_FALLING_EDGE	• Disable the PVM1 Extended Interrupt Rising & Falling Trigger. **Return value:** • None
__HAL_PWR_PVM1_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on selected EXTI line. **Return value:** • None
__HAL_PWR_PVM1_EXTI_GET_FLAG	**Description:** • Check whether the specified PVM1 EXTI interrupt flag is set or not. **Return value:** • EXTI: PVM1 Line Status.
__HAL_PWR_PVM1_EXTI_CLEAR_FLAG	**Description:** • Clear the PVM1 EXTI flag. **Return value:** • None
__HAL_PWR_PVM2_EXTI_ENABLE_IT	**Description:** • Enable the PVM2 Extended Interrupt Line. **Return value:** • None
__HAL_PWR_PVM2_EXTI_DISABLE_IT	**Description:** • Disable the PVM2 Extended Interrupt Line. **Return value:** • None
__HAL_PWR_PVM2_EXTI_ENABLE_EVENT	**Description:** • Enable the PVM2 Event Line. **Return value:** • None
__HAL_PWR_PVM2_EXTI_DISABLE_EVENT	**Description:** • Disable the PVM2 Event Line. **Return value:** • None
__HAL_PWR_PVM2_EXTI_ENABLE_RISING_EDGE	**Description:** • Enable the PVM2 Extended

Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVM2 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVM2 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVM2 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- PVM2 EXTI line configuration: set rising & falling edge trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVM2 Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None

__HAL_PWR_PVM2_EXTI_GET_FLAG

Description:

- Check whether the specified PVM2 EXTI interrupt flag is set or not.

Return value:

- EXTI: PVM2 Line Status.

__HAL_PWR_PVM2_EXTI_CLEAR_FLAG	**Description:** • Clear the PVM2 EXTI flag. **Return value:** • None
__HAL_PWR_PVM3_EXTI_ENABLE_IT	**Description:** • Enable the PVM3 Extended Interrupt Line. **Return value:** • None
__HAL_PWR_PVM3_EXTI_DISABLE_IT	**Description:** • Disable the PVM3 Extended Interrupt Line. **Return value:** • None
__HAL_PWR_PVM3_EXTI_ENABLE_EVENT	**Description:** • Enable the PVM3 Event Line. **Return value:** • None
__HAL_PWR_PVM3_EXTI_DISABLE_EVENT	**Description:** • Disable the PVM3 Event Line. **Return value:** • None
__HAL_PWR_PVM3_EXTI_ENABLE_RISING_EDGE	**Description:** • Enable the PVM3 Extended Interrupt Rising Trigger. **Return value:** • None
__HAL_PWR_PVM3_EXTI_DISABLE_RISING_EDGE	**Description:** • Disable the PVM3 Extended Interrupt Rising Trigger. **Return value:** • None
__HAL_PWR_PVM3_EXTI_ENABLE_FALLING_EDGE	**Description:** • Enable the PVM3 Extended Interrupt Falling Trigger. **Return value:** • None

__HAL_PWR_PVM3_EXTI_DISABLE_FALLING_EDGE	**Description:** • Disable the PVM3 Extended Interrupt Falling Trigger. **Return value:** • None
__HAL_PWR_PVM3_EXTI_ENABLE_RISING_FALLING_EDGE	**Description:** • PVM3 EXTI line configuration: set rising & falling edge trigger. **Return value:** • None
__HAL_PWR_PVM3_EXTI_DISABLE_RISING_FALLING_EDGE	**Description:** • Disable the PVM3 Extended Interrupt Rising & Falling Trigger. **Return value:** • None
__HAL_PWR_PVM3_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on selected EXTI line. **Return value:** • None
__HAL_PWR_PVM3_EXTI_GET_FLAG	**Description:** • Check whether the specified PVM3 EXTI interrupt flag is set or not. **Return value:** • EXTI: PVM3 Line Status.
__HAL_PWR_PVM3_EXTI_CLEAR_FLAG	**Description:** • Clear the PVM3 EXTI flag. **Return value:** • None
__HAL_PWR_PVM4_EXTI_ENABLE_IT	**Description:** • Enable the PVM4 Extended Interrupt Line. **Return value:** • None
__HAL_PWR_PVM4_EXTI_DISABLE_IT	**Description:** • Disable the PVM4 Extended Interrupt Line. **Return value:**

- None

__HAL_PWR_PVM4_EXTI_ENABLE_EVENT

Description:

- Enable the PVM4 Event Line.

Return value:

- None

__HAL_PWR_PVM4_EXTI_DISABLE_EVENT

Description:

- Disable the PVM4 Event Line.

Return value:

- None

__HAL_PWR_PVM4_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVM4 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVM4 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVM4 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVM4 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- PVM4 EXTI line configuration: set rising & falling edge trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVM4 Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None

__HAL_PWR_PVM4_EXTI_GET_FLAG

Description:

- Check whether or not the specified PVM4 EXTI interrupt flag is set.

Return value:

- EXTI: PVM4 Line Status.

__HAL_PWR_PVM4_EXTI_CLEAR_FLAG

Description:

- Clear the PVM4 EXTI flag.

Return value:

- None

__HAL_PWR_VOLTAGESCALING_CONFIG

Description:

- Configure the main internal regulator output voltage.

Parameters:

- __REGULATOR__: specifies the regulator output voltage to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values:
  - PWR_REGULATOR_VOLTAGE_SCALE1 Regulator voltage output range 1 mode, typical output voltage at 1.2 V, system frequency up to 80 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE2 Regulator voltage output range 2 mode, typical output voltage at 1.0 V, system frequency up to 26 MHz.

Return value:

- None

Notes:

- This macro is similar to HAL_PWREx_ControlVoltageScaling() API but doesn't check whether or not VOSF flag is cleared when moving from range 2 to range 1. User may resort to __HAL_PWR_GET_FLAG() macro to check VOSF bit resetting.

PWR Status Flags

PWR_FLAG_WUF1	Wakeup event on wakeup pin 1
PWR_FLAG_WUF2	Wakeup event on wakeup pin 2
PWR_FLAG_WUF3	Wakeup event on wakeup pin 3
PWR_FLAG_WUF4	Wakeup event on wakeup pin 4
PWR_FLAG_WUF5	Wakeup event on wakeup pin 5
PWR_FLAG_WU	Encompass wakeup event on all wakeup pins
PWR_FLAG_SB	Standby flag
PWR_FLAG_WUFI	Wakeup on internal wakeup line
PWR_FLAG_REGLPS	Low-power regulator start flag
PWR_FLAG_REGLPF	Low-power regulator flag
PWR_FLAG_VOSF	Voltage scaling flag
PWR_FLAG_PVDO	Power Voltage Detector output flag
PWR_FLAG_PVMO1	Power Voltage Monitoring 1 output flag
PWR_FLAG_PVMO2	Power Voltage Monitoring 2 output flag
PWR_FLAG_PVMO3	Power Voltage Monitoring 3 output flag
PWR_FLAG_PVMO4	Power Voltage Monitoring 4 output flag

GPIO port

PWR_GPIO_A	GPIO port A
PWR_GPIO_B	GPIO port B
PWR_GPIO_C	GPIO port C
PWR_GPIO_D	GPIO port D
PWR_GPIO_E	GPIO port E
PWR_GPIO_F	GPIO port F
PWR_GPIO_G	GPIO port G
PWR_GPIO_H	GPIO port H
PWR_GPIO_I	GPIO port I

GPIO bit number for I/O setting in standby/shutdown mode

PWR_GPIO_BIT_0	GPIO port I/O pin 0
PWR_GPIO_BIT_1	GPIO port I/O pin 1
PWR_GPIO_BIT_2	GPIO port I/O pin 2
PWR_GPIO_BIT_3	GPIO port I/O pin 3
PWR_GPIO_BIT_4	GPIO port I/O pin 4
PWR_GPIO_BIT_5	GPIO port I/O pin 5
PWR_GPIO_BIT_6	GPIO port I/O pin 6

PWR_GPIO_BIT_7	GPIO port I/O pin 7
PWR_GPIO_BIT_8	GPIO port I/O pin 8
PWR_GPIO_BIT_9	GPIO port I/O pin 9
PWR_GPIO_BIT_10	GPIO port I/O pin 10
PWR_GPIO_BIT_11	GPIO port I/O pin 11
PWR_GPIO_BIT_12	GPIO port I/O pin 12
PWR_GPIO_BIT_13	GPIO port I/O pin 13
PWR_GPIO_BIT_14	GPIO port I/O pin 14
PWR_GPIO_BIT_15	GPIO port I/O pin 15

PWR PVM event lines

PWR_EVENT_LINE_PVM1	Event line 35 Connected to the PVM1 EXTI Line
PWR_EVENT_LINE_PVM2	Event line 36 Connected to the PVM2 EXTI Line
PWR_EVENT_LINE_PVM3	Event line 37 Connected to the PVM3 EXTI Line
PWR_EVENT_LINE_PVM4	Event line 38 Connected to the PVM4 EXTI Line

PWR PVM external interrupts lines

PWR_EXTI_LINE_PVM1	External interrupt line 35 Connected to the PVM1 EXTI Line
PWR_EXTI_LINE_PVM2	External interrupt line 36 Connected to the PVM2 EXTI Line
PWR_EXTI_LINE_PVM3	External interrupt line 37 Connected to the PVM3 EXTI Line
PWR_EXTI_LINE_PVM4	External interrupt line 38 Connected to the PVM4 EXTI Line

PWR PVM interrupt and event mode

PWR_PVM_MODE_NORMAL	basic mode is used
PWR_PVM_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
PWR_PVM_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
PWR_PVM_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
PWR_PVM_MODE_EVENT_RISING	Event Mode with Rising edge trigger detection
PWR_PVM_MODE_EVENT_FALLING	Event Mode with Falling edge trigger detection
PWR_PVM_MODE_EVENT_RISING_FALLING	Event Mode with Rising/Falling edge trigger detection

PWR PVM Mode Mask

PVM_MODE_IT	Mask for interruption yielded by PVM threshold crossing
PVM_MODE_EVT	Mask for event yielded by PVM threshold crossing
PVM_RISING_EDGE	Mask for rising edge set as PVM trigger
PVM_FALLING_EDGE	Mask for falling edge set as PVM trigger

Peripheral Voltage Monitoring type

PWR_PVM_1	Peripheral Voltage Monitoring 1 enable: VDDUSB versus 1.2 V (applicable when USB feature is supported)
PWR_PVM_2	Peripheral Voltage Monitoring 2 enable: VDDIO2 versus 0.9 V (applicable when VDDIO2 is present on device)
PWR_PVM_3	Peripheral Voltage Monitoring 3 enable: VDDA versus 1.62 V
PWR_PVM_4	Peripheral Voltage Monitoring 4 enable: VDDA versus 2.2 V

PWR Regulator voltage scale

PWR_REGULATOR_VOLTAGE_SCALE1_BOOST	Voltage scaling range 1 boost mode
PWR_REGULATOR_VOLTAGE_SCALE1	Voltage scaling range 1 normal mode
PWR_REGULATOR_VOLTAGE_SCALE2	Voltage scaling range 2

PWR Extended Flag Setting Time Out Value

PWR_FLAG_SETTING_DELAY_US	Time out value for REGLPF and VOSF flags setting

PWR battery charging

PWR_BATTERY_CHARGING_DISABLE

PWR_BATTERY_CHARGING_ENABLE

PWR battery charging resistor selection

PWR_BATTERY_CHARGING_RESISTOR_5	VBAT charging through a 5 kOhms resistor
PWR_BATTERY_CHARGING_RESISTOR_1_5	VBAT charging through a 1.5 kOhms resistor

PWR wake-up pins

PWR_WAKEUP_PIN1	Wakeup pin 1 (with high level polarity)
PWR_WAKEUP_PIN2	Wakeup pin 2 (with high level polarity)
PWR_WAKEUP_PIN3	Wakeup pin 3 (with high level polarity)
PWR_WAKEUP_PIN4	Wakeup pin 4 (with high level polarity)
PWR_WAKEUP_PIN5	Wakeup pin 5 (with high level polarity)
PWR_WAKEUP_PIN1_HIGH	Wakeup pin 1 (with high level polarity)
PWR_WAKEUP_PIN2_HIGH	Wakeup pin 2 (with high level polarity)
PWR_WAKEUP_PIN3_HIGH	Wakeup pin 3 (with high level polarity)
PWR_WAKEUP_PIN4_HIGH	Wakeup pin 4 (with high level polarity)
PWR_WAKEUP_PIN5_HIGH	Wakeup pin 5 (with high level polarity)
PWR_WAKEUP_PIN1_LOW	Wakeup pin 1 (with low level polarity)
PWR_WAKEUP_PIN2_LOW	Wakeup pin 2 (with low level polarity)
PWR_WAKEUP_PIN3_LOW	Wakeup pin 3 (with low level polarity)
PWR_WAKEUP_PIN4_LOW	Wakeup pin 4 (with low level polarity)
PWR_WAKEUP_PIN5_LOW	Wakeup pin 5 (with low level polarity)

Shift to apply to retrieve polarity information from PWR_WAKEUP_PINy_xxx constants

PWR_WUP_POLARITY_SHIFT	Internal constant used to retrieve wakeup pin polariry

51 HAL RCC Generic Driver

51.1 RCC Firmware driver registers structures

51.1.1 RCC_PLLInitTypeDef

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLP***
- ***uint32_t PLLQ***
- ***uint32_t PLLR***

Field Documentation

- ***uint32_t RCC_PLLInitTypeDef::PLLState***
  The new state of the PLL. This parameter can be a value of ***RCC_PLL_Config***
- ***uint32_t RCC_PLLInitTypeDef::PLLSource***
  RCC_PLLSource: PLL entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLLInitTypeDef::PLLM***
  PLLM: Division factor for PLL VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16 on STM32L4Rx/STM32L4Sx devices. This parameter must be a number between Min_Data = 1 and Max_Data = 8 on the other devices
- ***uint32_t RCC_PLLInitTypeDef::PLLN***
  PLLN: Multiplication factor for PLL VCO output clock. This parameter must be a number between Min_Data = 8 and Max_Data = 86
- ***uint32_t RCC_PLLInitTypeDef::PLLP***
  PLLP: Division factor for SAI clock. This parameter must be a value of ***RCC_PLLP_Clock_Divider***
- ***uint32_t RCC_PLLInitTypeDef::PLLQ***
  PLLQ: Division factor for SDMMC1, RNG and USB clocks. This parameter must be a value of ***RCC_PLLQ_Clock_Divider***
- ***uint32_t RCC_PLLInitTypeDef::PLLR***
  PLLR: Division for the main system clock. User have to set the PLLR parameter correctly to not exceed max frequency 80MHZ. This parameter must be a value of ***RCC_PLLR_Clock_Divider***

51.1.2 RCC_OscInitTypeDef

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t LSEState***
- ***uint32_t HSIState***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSIState***
- ***uint32_t MSIState***
- ***uint32_t MSICalibrationValue***

- ***uint32_t MSIClockRange***
- ***uint32_t HSI48State***
- ***RCC_PLLInitTypeDef PLL***

Field Documentation

- ***uint32_t RCC_OscInitTypeDef::OscillatorType***
  The oscillators to be configured. This parameter can be a value of ***RCC_Oscillator_Type***
- ***uint32_t RCC_OscInitTypeDef::HSEState***
  The new state of the HSE. This parameter can be a value of ***RCC_HSE_Config***
- ***uint32_t RCC_OscInitTypeDef::LSEState***
  The new state of the LSE. This parameter can be a value of ***RCC_LSE_Config***
- ***uint32_t RCC_OscInitTypeDef::HSIState***
  The new state of the HSI. This parameter can be a value of ***RCC_HSI_Config***
- ***uint32_t RCC_OscInitTypeDef::HSICalibrationValue***
  The calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F on STM32L43x/STM32L44x/STM32L47x/STM32L48x devices. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F on the other devices
- ***uint32_t RCC_OscInitTypeDef::LSIState***
  The new state of the LSI. This parameter can be a value of ***RCC_LSI_Config***
- ***uint32_t RCC_OscInitTypeDef::MSIState***
  The new state of the MSI. This parameter can be a value of ***RCC_MSI_Config***
- ***uint32_t RCC_OscInitTypeDef::MSICalibrationValue***
  The calibration trimming value (default is RCC_MSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t RCC_OscInitTypeDef::MSIClockRange***
  The MSI frequency range. This parameter can be a value of ***RCC_MSI_Clock_Range***
- ***uint32_t RCC_OscInitTypeDef::HSI48State***
  The new state of the HSI48 (only applicable to STM32L43x/STM32L44x/STM32L49x/STM32L4Ax devices). This parameter can be a value of ***RCC_HSI48_Config***
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL***
  Main PLL structure parameters

51.1.3 RCC_ClkInitTypeDef

Data Fields

- ***uint32_t ClockType***
- ***uint32_t SYSCLKSource***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***

Field Documentation

- ***uint32_t RCC_ClkInitTypeDef::ClockType***
  The clock to be configured. This parameter can be a value of ***RCC_System_Clock_Type***
- ***uint32_t RCC_ClkInitTypeDef::SYSCLKSource***
  The clock source used as system clock (SYSCLK). This parameter can be a value of ***RCC_System_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_AHB_Clock_Source***

- ***uint32_t RCC_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_Clock_Source***

51.2 RCC Firmware driver API description

51.2.1 RCC specific features

After reset the device is running from Multiple Speed Internal oscillator (4 MHz) with Flash 0 wait state. Flash prefetch buffer, D-Cache and I-Cache are disabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHBs) and Low speed (APBs) busses: all peripherals mapped on these busses are running at MSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in analog mode, except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB busses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals which clocks are not derived from the System clock (SAIx, RTC, ADC, USB OTG FS/SDMMC1/RNG)

51.2.2 Initialization and de-initialization functions

This section provides functions allowing to configure the internal and external oscillators (HSE, HSI, LSE, MSI, LSI, PLL, CSS and MCO) and the System busses clocks (SYSCLK, AHB, APB1 and APB2).

Internal/external clock and PLL configuration

- HSI (high-speed internal): 16 MHz factory-trimmed RC used directly or through the PLL as System clock source.
- MSI (Mutiple Speed Internal): Its frequency is software trimmable from 100KHZ to 48MHZ. It can be used to generate the clock for the USB OTG FS (48 MHz). The number of flash wait states is automatically adjusted when MSI range is updated with HAL_RCC_OscConfig() and the MSI is used as System clock source.
- LSI (low-speed internal): 32 KHz low consumption RC used as IWDG and/or RTC clock source.
- HSE (high-speed external): 4 to 48 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also optionally as RTC clock source.
- LSE (low-speed external): 32.768 KHz oscillator used optionally as RTC clock source.
- PLL (clocked by HSI, HSE or MSI) providing up to three independent output clocks:
  - The first output is used to generate the high speed system clock (up to 80MHz).
  - The second output is used to generate the clock for the USB OTG FS (48 MHz), the random analog generator (<=48 MHz) and the SDMMC1 (<= 48 MHz).
  - The third output is used to generate an accurate clock to achieve high-quality audio performance on SAI interface.

- PLLSAI1 (clocked by HSI, HSE or MSI) providing up to three independent output clocks:
  - The first output is used to generate SAR ADC1 clock.
  - The second output is used to generate the clock for the USB OTG FS (48 MHz), the random analog generator (<=48 MHz) and the SDMMC1 (<= 48 MHz).
  - The Third output is used to generate an accurate clock to achieve high-quality audio performance on SAI interface.
- PLLSAI2 (clocked by HSI , HSE or MSI) providing up to two independent output clocks:
  - The first output is used to generate SAR ADC2 clock.
  - The second output is used to generate an accurate clock to achieve high-quality audio performance on SAI interface.
- CSS (Clock security system): once enabled, if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M4 NMI (Non-Maskable Interrupt) exception vector.
- MCO (microcontroller clock output): used to output MSI, LSI, HSI, LSE, HSE or main PLL clock (through a configurable prescaler) on PA8 pin.

System, AHB and APB busses clocks configuration

- Several clock sources can be used to drive the System clock (SYSCLK): MSI, HSI, HSE and main PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...). APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "HAL_RCC_GetSysClockFreq()" function to retrieve the frequencies of these clocks. All the peripheral clocks are derived from the System clock (SYSCLK) except: SAI: the SAI clock can be derived either from a specific PLL (PLLSAI1) or (PLLSAI2) or from an external clock mapped on the SAI_CKIN pin. You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 31. You have to use __HAL_RCC_RTC_ENABLE() and HAL_RCCEx_PeriphCLKConfig() function to configure this clock.USB OTG FS, SDMMC1 and RNG: USB OTG FS requires a frequency equal to 48 MHz to work correctly, while the SDMMC1 and RNG peripherals require a frequency equal or lower than to 48 MHz. This clock is derived of the main PLL or PLLSAI1 through PLLQ divider. You have to enable the peripheral clock and use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.IWDG clock which is always the LSI clock.
- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 80 MHz. The clock source frequency should be adapted depending on the device voltage range as listed in the Reference Manual "Clock source frequency versus voltage scaling" chapter.

This section contains the following APIs:

- ***HAL_RCC_DeInit()***
- ***HAL_RCC_OscConfig()***
- ***HAL_RCC_ClockConfig()***

51.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to:

- Ouput clock to MCO pin.
- Retrieve current clock frequencies.

- Enable the Clock Security System.

This section contains the following APIs:

- *HAL_RCC_MCOConfig()*
- *HAL_RCC_GetSysClockFreq()*
- *HAL_RCC_GetHCLKFreq()*
- *HAL_RCC_GetPCLK1Freq()*
- *HAL_RCC_GetPCLK2Freq()*
- *HAL_RCC_GetOscConfig()*
- *HAL_RCC_GetClockConfig()*
- *HAL_RCC_EnableCSS()*
- *HAL_RCC_NMI_IRQHandler()*
- *HAL_RCC_CSSCallback()*

51.2.4 Detailed description of functions

HAL_RCC_DeInit

Function name	**void HAL_RCC_DeInit (void)**
Function description	Reset the RCC clock configuration to the default reset state.
Return values	• **None:**
Notes	• The default reset state of the clock configuration is given below: MSI ON and used as system clock sourceHSE, HSI, PLL, PLLSAI1 and PLLISAI2 OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO1 OFFAll interrupts disabled • This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

HAL_RCC_OscConfig

Function name	**HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)**
Function description	Initialize the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.
Parameters	• **RCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.
Return values	• **HAL:** status
Notes	• The PLL is not disabled when used as system clock. • Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. • Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass.

HAL_RCC_ClockConfig

Function name	**HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)**

Function description	Initialize the CPU, AHB and APB busses clocks according to the specified parameters in the RCC_ClkInitStruct.
Parameters	• **RCC_ClkInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral. • **FLatency:** FLASH Latency This parameter can be one of the following values: – FLASH_LATENCY_0 FLASH 0 Latency cycle – FLASH_LATENCY_1 FLASH 1 Latency cycle – FLASH_LATENCY_2 FLASH 2 Latency cycles – FLASH_LATENCY_3 FLASH 3 Latency cycles – FLASH_LATENCY_4 FLASH 4 Latency cycles – FLASH_LATENCY_5 FLASH 5 Latency cycles – FLASH_LATENCY_6 FLASH 6 Latency cycles – FLASH_LATENCY_7 FLASH 7 Latency cycles – FLASH_LATENCY_8 FLASH 8 Latency cycles – FLASH_LATENCY_9 FLASH 9 Latency cycles – FLASH_LATENCY_10 FLASH 10 Latency cycles – FLASH_LATENCY_11 FLASH 11 Latency cycles – FLASH_LATENCY_12 FLASH 12 Latency cycles – FLASH_LATENCY_13 FLASH 13 Latency cycles – FLASH_LATENCY_14 FLASH 14 Latency cycles – FLASH_LATENCY_15 FLASH 15 Latency cycles
Return values	• **None:**
Notes	• The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function • The MSI is used by default as system clock source after startup from Reset, wake-up from STANDBY mode. After restart from Reset, the MSI frequency is set to its default value 4 MHz. • The HSI can be selected as system clock source after from STOP modes or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). • A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source is ready. • You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source. • Depending on the device voltage range, the software has to set correctly HPRE[3:0] bits to ensure that HCLK not exceed the maximum allowed frequency (for more details refer to section above "Initialization/de-initialization functions")

HAL_RCC_MCOConfig

Function name	**void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)**
Function description	Select the clock source to output on MCO pin(PA8).

Parameters	• **RCC_MCOx:** specifies the output direction for the clock source. For STM32L4xx family this parameter can have only one value: – RCC_MCO1 Clock source to output on MCO1 pin(PA8). • **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values: – RCC_MCO1SOURCE_NOCLOCK MCO output disabled, no clock on MCO – RCC_MCO1SOURCE_SYSCLK system clock selected as MCO source – RCC_MCO1SOURCE_MSI MSI clock selected as MCO source – RCC_MCO1SOURCE_HSI HSI clock selected as MCO source – RCC_MCO1SOURCE_HSE HSE clock selected as MCO sourcee – RCC_MCO1SOURCE_PLLCLK main PLL clock selected as MCO source – RCC_MCO1SOURCE_LSI LSI clock selected as MCO source – RCC_MCO1SOURCE_LSE LSE clock selected as MCO source • **RCC_MCODiv:** specifies the MCO prescaler. This parameter can be one of the following values: – RCC_MCODIV_1 no division applied to MCO clock – RCC_MCODIV_2 division by 2 applied to MCO clock – RCC_MCODIV_4 division by 4 applied to MCO clock – RCC_MCODIV_8 division by 8 applied to MCO clock – RCC_MCODIV_16 division by 16 applied to MCO clock
Return values	• **None:**
Notes	• PA8 should be configured in alternate function mode.

HAL_RCC_EnableCSS

Function name	**void HAL_RCC_EnableCSS (void)**
Function description	Enable the Clock Security System.
Return values	• **None:**
Notes	• If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M4 NMI (Non-Maskable Interrupt) exception vector. • The Clock Security System can only be cleared by reset.

HAL_RCC_GetSysClockFreq

Function name	**uint32_t HAL_RCC_GetSysClockFreq (void)**
Function description	Return the SYSCLK frequency.
Return values	• **SYSCLK:** frequency

Notes	• The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source: • If SYSCLK source is MSI, function returns values based on MSI Value as defined by the MSI range. • If SYSCLK source is HSI, function returns values based on HSI_VALUE(*) • If SYSCLK source is HSE, function returns values based on HSE_VALUE(**) • If SYSCLK source is PLL, function returns values based on HSE_VALUE(**), HSI_VALUE(*) or MSI Value multiplied/divided by the PLL factors. • (*) HSI_VALUE is a constant defined in stm32l4xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature. • (**) HSE_VALUE is a constant defined in stm32l4xx_hal_conf.h file (default value 8 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result. • The result of this function could be not correct when using fractional value for HSE crystal. • This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters. • Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetHCLKFreq

Function name	**uint32_t HAL_RCC_GetHCLKFreq (void)**
Function description	Return the HCLK frequency.
Return values	• **HCLK:** frequency in Hz
Notes	• Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect. • The SystemCoreClock CMSIS variable is used to store System Clock Frequency.

HAL_RCC_GetPCLK1Freq

Function name	**uint32_t HAL_RCC_GetPCLK1Freq (void)**
Function description	Return the PCLK1 frequency.
Return values	• **PCLK1:** frequency in Hz
Notes	• Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK2Freq

Function name	**uint32_t HAL_RCC_GetPCLK2Freq (void)**
Function description	Return the PCLK2 frequency.
Return values	• **PCLK2:** frequency in Hz
Notes	• Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetOscConfig

Function name	**void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)**
Function description	Configure the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	• **RCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that will be configured.
Return values	• **None:**

HAL_RCC_GetClockConfig

Function name	**void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)**
Function description	Configure the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	• **RCC_ClkInitStruct:** pointer to an RCC_ClkInitTypeDef structure that will be configured. • **pFLatency:** Pointer on the Flash Latency.
Return values	• **None:**

HAL_RCC_NMI_IRQHandler

Function name	**void HAL_RCC_NMI_IRQHandler (void)**
Function description	Handle the RCC Clock Security System interrupt request.
Return values	• **None:**
Notes	• This API should be called under the NMI_Handler().

HAL_RCC_CSSCallback

Function name	**void HAL_RCC_CSSCallback (void)**
Function description	RCC Clock Security System interrupt callback.
Return values	• **none:**

51.3 RCC Firmware driver defines

51.3.1 RCC

AHB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_DMA1_CLK_SLEEP_ENABLE

__HAL_RCC_DMA2_CLK_SLEEP_ENABLE

__HAL_RCC_DMAMUX1_CLK_SLEEP_ENABLE

__HAL_RCC_FLASH_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_CRC_CLK_SLEEP_ENABLE

__HAL_RCC_TSC_CLK_SLEEP_ENABLE

__HAL_RCC_DMA2D_CLK_SLEEP_ENABLE

__HAL_RCC_GFXMMU_CLK_SLEEP_ENABLE

__HAL_RCC_DMA1_CLK_SLEEP_DISABLE

__HAL_RCC_DMA2_CLK_SLEEP_DISABLE

__HAL_RCC_DMAMUX1_CLK_SLEEP_DISABLE

__HAL_RCC_FLASH_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_CRC_CLK_SLEEP_DISABLE

__HAL_RCC_TSC_CLK_SLEEP_DISABLE

__HAL_RCC_DMA2D_CLK_SLEEP_DISABLE

__HAL_RCC_GFXMMU_CLK_SLEEP_DISABLE

AHB1 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_DMA1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DMA2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DMAMUX1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FLASH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SRAM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CRC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TSC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DMA2D_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GFXMMU_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DMA1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DMA2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DMAMUX1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_FLASH_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SRAM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CRC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TSC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DMA2D_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GFXMMU_IS_CLK_SLEEP_DISABLED

AHB1 Peripheral Force Release Reset

__HAL_RCC_AHB1_FORCE_RESET

__HAL_RCC_DMA1_FORCE_RESET

__HAL_RCC_DMA2_FORCE_RESET

__HAL_RCC_DMAMUX1_FORCE_RESET

__HAL_RCC_FLASH_FORCE_RESET

__HAL_RCC_CRC_FORCE_RESET

__HAL_RCC_TSC_FORCE_RESET

__HAL_RCC_DMA2D_FORCE_RESET

__HAL_RCC_GFXMMU_FORCE_RESET

__HAL_RCC_AHB1_RELEASE_RESET

__HAL_RCC_DMA1_RELEASE_RESET

__HAL_RCC_DMA2_RELEASE_RESET

__HAL_RCC_DMAMUX1_RELEASE_RESET

__HAL_RCC_FLASH_RELEASE_RESET

__HAL_RCC_CRC_RELEASE_RESET

__HAL_RCC_TSC_RELEASE_RESET

__HAL_RCC_DMA2D_RELEASE_RESET

__HAL_RCC_GFXMMU_RELEASE_RESET

AHB1 Peripheral Clock Enable Disable

__HAL_RCC_DMA1_CLK_ENABLE

__HAL_RCC_DMA2_CLK_ENABLE

__HAL_RCC_DMAMUX1_CLK_ENABLE

__HAL_RCC_FLASH_CLK_ENABLE

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_TSC_CLK_ENABLE

__HAL_RCC_DMA2D_CLK_ENABLE

__HAL_RCC_GFXMMU_CLK_ENABLE

__HAL_RCC_DMA1_CLK_DISABLE

__HAL_RCC_DMA2_CLK_DISABLE

__HAL_RCC_DMAMUX1_CLK_DISABLE

__HAL_RCC_FLASH_CLK_DISABLE

__HAL_RCC_CRC_CLK_DISABLE

__HAL_RCC_TSC_CLK_DISABLE

__HAL_RCC_DMA2D_CLK_DISABLE

__HAL_RCC_GFXMMU_CLK_DISABLE

AHB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_DMA1_IS_CLK_ENABLED

__HAL_RCC_DMA2_IS_CLK_ENABLED

__HAL_RCC_DMAMUX1_IS_CLK_ENABLED

__HAL_RCC_FLASH_IS_CLK_ENABLED

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_TSC_IS_CLK_ENABLED

__HAL_RCC_DMA2D_IS_CLK_ENABLED

__HAL_RCC_GFXMMU_IS_CLK_ENABLED

__HAL_RCC_DMA1_IS_CLK_DISABLED

__HAL_RCC_DMA2_IS_CLK_DISABLED

__HAL_RCC_DMAMUX1_IS_CLK_DISABLED

__HAL_RCC_FLASH_IS_CLK_DISABLED

__HAL_RCC_CRC_IS_CLK_DISABLED

__HAL_RCC_TSC_IS_CLK_DISABLED

__HAL_RCC_DMA2D_IS_CLK_DISABLED

__HAL_RCC_GFXMMU_IS_CLK_DISABLED

AHB2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_ENABLED

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_GPIOC_IS_CLK_ENABLED

__HAL_RCC_GPIOD_IS_CLK_ENABLED

__HAL_RCC_GPIOE_IS_CLK_ENABLED

__HAL_RCC_GPIOF_IS_CLK_ENABLED

__HAL_RCC_GPIOG_IS_CLK_ENABLED

__HAL_RCC_GPIOH_IS_CLK_ENABLED

__HAL_RCC_GPIOI_IS_CLK_ENABLED

__HAL_RCC_USB_OTG_FS_IS_CLK_ENABLED

__HAL_RCC_ADC_IS_CLK_ENABLED

__HAL_RCC_DCMI_IS_CLK_ENABLED

__HAL_RCC_AES_IS_CLK_ENABLED

__HAL_RCC_HASH_IS_CLK_ENABLED

__HAL_RCC_RNG_IS_CLK_ENABLED

__HAL_RCC_GPIOA_IS_CLK_DISABLED

__HAL_RCC_GPIOB_IS_CLK_DISABLED

__HAL_RCC_GPIOC_IS_CLK_DISABLED

__HAL_RCC_GPIOD_IS_CLK_DISABLED

__HAL_RCC_GPIOE_IS_CLK_DISABLED

__HAL_RCC_GPIOF_IS_CLK_DISABLED

__HAL_RCC_GPIOG_IS_CLK_DISABLED

__HAL_RCC_GPIOH_IS_CLK_DISABLED

__HAL_RCC_GPIOI_IS_CLK_DISABLED

__HAL_RCC_USB_OTG_FS_IS_CLK_DISABLED

__HAL_RCC_ADC_IS_CLK_DISABLED

__HAL_RCC_DCMI_IS_CLK_DISABLED

__HAL_RCC_AES_IS_CLK_DISABLED

__HAL_RCC_HASH_IS_CLK_DISABLED

__HAL_RCC_RNG_IS_CLK_DISABLED

AHB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPIOA_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOB_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOC_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOD_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOE_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOF_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOG_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOH_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOI_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM2_CLK_SLEEP_ENABLE

__HAL_RCC_USB_OTG_FS_CLK_SLEEP_ENABLE

__HAL_RCC_ADC_CLK_SLEEP_ENABLE

__HAL_RCC_DCMI_CLK_SLEEP_ENABLE

__HAL_RCC_AES_CLK_SLEEP_ENABLE

__HAL_RCC_HASH_CLK_SLEEP_ENABLE

__HAL_RCC_RNG_CLK_SLEEP_ENABLE

__HAL_RCC_OSPIM_CLK_SLEEP_ENABLE

__HAL_RCC_SDMMC1_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOA_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOB_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOC_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOD_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOE_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOF_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOG_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOH_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOI_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM2_CLK_SLEEP_DISABLE

__HAL_RCC_USB_OTG_FS_CLK_SLEEP_DISABLE

__HAL_RCC_ADC_CLK_SLEEP_DISABLE

__HAL_RCC_DCMI_CLK_SLEEP_DISABLE

__HAL_RCC_AES_CLK_SLEEP_DISABLE

__HAL_RCC_HASH_CLK_SLEEP_DISABLE

__HAL_RCC_RNG_CLK_SLEEP_DISABLE

__HAL_RCC_OSPIM_CLK_SLEEP_DISABLE

__HAL_RCC_SDMMC1_CLK_SLEEP_DISABLE

AHB2 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOD_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOE_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOF_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SRAM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB_OTG_FS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ADC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DCMI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_AES_IS_CLK_SLEEP_ENABLED

__HAL_RCC_HASH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RNG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_OSPIM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SDMMC1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOA_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOD_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOE_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOF_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOH_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOI_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SRAM2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USB_OTG_FS_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ADC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DCMI_IS_CLK_SLEEP_DISABLED

__HAL_RCC_AES_IS_CLK_SLEEP_DISABLED

__HAL_RCC_HASH_IS_CLK_SLEEP_DISABLED

__HAL_RCC_RNG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_OSPIM_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SDMMC1_IS_CLK_SLEEP_DISABLED

AHB2 Peripheral Force Release Reset

__HAL_RCC_AHB2_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_GPIOE_FORCE_RESET

__HAL_RCC_GPIOF_FORCE_RESET

__HAL_RCC_GPIOG_FORCE_RESET

__HAL_RCC_GPIOH_FORCE_RESET

__HAL_RCC_GPIOI_FORCE_RESET

__HAL_RCC_USB_OTG_FS_FORCE_RESET

__HAL_RCC_ADC_FORCE_RESET

__HAL_RCC_DCMI_FORCE_RESET

__HAL_RCC_AES_FORCE_RESET

__HAL_RCC_HASH_FORCE_RESET

__HAL_RCC_RNG_FORCE_RESET

__HAL_RCC_OSPIM_FORCE_RESET

__HAL_RCC_SDMMC1_FORCE_RESET

__HAL_RCC_AHB2_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

__HAL_RCC_GPIOE_RELEASE_RESET

__HAL_RCC_GPIOF_RELEASE_RESET

__HAL_RCC_GPIOG_RELEASE_RESET

__HAL_RCC_GPIOH_RELEASE_RESET

__HAL_RCC_GPIOI_RELEASE_RESET

__HAL_RCC_USB_OTG_FS_RELEASE_RESET

__HAL_RCC_ADC_RELEASE_RESET

__HAL_RCC_DCMI_RELEASE_RESET

__HAL_RCC_AES_RELEASE_RESET

__HAL_RCC_HASH_RELEASE_RESET

__HAL_RCC_RNG_RELEASE_RESET

__HAL_RCC_OSPIM_RELEASE_RESET

__HAL_RCC_SDMMC1_RELEASE_RESET

AHB2 Peripheral Clock Enable Disable

__HAL_RCC_GPIOA_CLK_ENABLE

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_GPIOC_CLK_ENABLE

__HAL_RCC_GPIOD_CLK_ENABLE

__HAL_RCC_GPIOE_CLK_ENABLE

__HAL_RCC_GPIOF_CLK_ENABLE

__HAL_RCC_GPIOG_CLK_ENABLE

__HAL_RCC_GPIOH_CLK_ENABLE

__HAL_RCC_GPIOI_CLK_ENABLE

__HAL_RCC_USB_OTG_FS_CLK_ENABLE

__HAL_RCC_ADC_CLK_ENABLE

__HAL_RCC_DCMI_CLK_ENABLE

__HAL_RCC_AES_CLK_ENABLE

__HAL_RCC_HASH_CLK_ENABLE

__HAL_RCC_RNG_CLK_ENABLE

__HAL_RCC_OSPIM_CLK_ENABLE

__HAL_RCC_SDMMC1_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOD_CLK_DISABLE

__HAL_RCC_GPIOE_CLK_DISABLE

__HAL_RCC_GPIOF_CLK_DISABLE

__HAL_RCC_GPIOG_CLK_DISABLE

__HAL_RCC_GPIOH_CLK_DISABLE

__HAL_RCC_GPIOI_CLK_DISABLE

__HAL_RCC_USB_OTG_FS_CLK_DISABLE

__HAL_RCC_ADC_CLK_DISABLE

__HAL_RCC_DCMI_CLK_DISABLE

__HAL_RCC_AES_CLK_DISABLE

__HAL_RCC_HASH_CLK_DISABLE

__HAL_RCC_RNG_CLK_DISABLE

__HAL_RCC_OSPIM_CLK_DISABLE

__HAL_RCC_SDMMC1_CLK_DISABLE

AHB3 Peripheral Clock Enable Disable

__HAL_RCC_FMC_CLK_ENABLE

__HAL_RCC_OSPI1_CLK_ENABLE

__HAL_RCC_OSPI2_CLK_ENABLE

__HAL_RCC_FMC_CLK_DISABLE

__HAL_RCC_OSPI1_CLK_DISABLE

__HAL_RCC_OSPI2_CLK_DISABLE

AHB3 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_FMC_IS_CLK_ENABLED

__HAL_RCC_FMC_IS_CLK_DISABLED

AHB3 Peripheral Clock Sleep Enable Disable

__HAL_RCC_OSPI1_CLK_SLEEP_ENABLE

__HAL_RCC_OSPI2_CLK_SLEEP_ENABLE

__HAL_RCC_FMC_CLK_SLEEP_ENABLE

__HAL_RCC_OSPI1_CLK_SLEEP_DISABLE

__HAL_RCC_OSPI2_CLK_SLEEP_DISABLE

__HAL_RCC_FMC_CLK_SLEEP_DISABLE

AHB3 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_OSPI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_OSPI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FMC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_OSPI1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_OSPI2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_FMC_IS_CLK_SLEEP_DISABLED

AHB3 Peripheral Force Release Reset

__HAL_RCC_AHB3_FORCE_RESET

__HAL_RCC_FMC_FORCE_RESET

__HAL_RCC_OSPI1_FORCE_RESET

__HAL_RCC_OSPI2_FORCE_RESET

__HAL_RCC_AHB3_RELEASE_RESET

__HAL_RCC_FMC_RELEASE_RESET

__HAL_RCC_OSPI1_RELEASE_RESET

__HAL_RCC_OSPI2_RELEASE_RESET

AHB Clock Source

RCC_SYSCLK_DIV1	SYSCLK not divided
RCC_SYSCLK_DIV2	SYSCLK divided by 2
RCC_SYSCLK_DIV4	SYSCLK divided by 4
RCC_SYSCLK_DIV8	SYSCLK divided by 8
RCC_SYSCLK_DIV16	SYSCLK divided by 16
RCC_SYSCLK_DIV64	SYSCLK divided by 64
RCC_SYSCLK_DIV128	SYSCLK divided by 128
RCC_SYSCLK_DIV256	SYSCLK divided by 256
RCC_SYSCLK_DIV512	SYSCLK divided by 512

APB1 APB2 Clock Source

RCC_HCLK_DIV1	HCLK not divided
RCC_HCLK_DIV2	HCLK divided by 2
RCC_HCLK_DIV4	HCLK divided by 4
RCC_HCLK_DIV8	HCLK divided by 8
RCC_HCLK_DIV16	HCLK divided by 16

APB1 Peripheral Clock Enable Disable

__HAL_RCC_TIM2_CLK_ENABLE

__HAL_RCC_TIM3_CLK_ENABLE

__HAL_RCC_TIM4_CLK_ENABLE

__HAL_RCC_TIM5_CLK_ENABLE

__HAL_RCC_TIM6_CLK_ENABLE

__HAL_RCC_TIM7_CLK_ENABLE

__HAL_RCC_RTCAPB_CLK_ENABLE

__HAL_RCC_WWDG_CLK_ENABLE

__HAL_RCC_SPI2_CLK_ENABLE

__HAL_RCC_SPI3_CLK_ENABLE

__HAL_RCC_USART2_CLK_ENABLE

__HAL_RCC_USART3_CLK_ENABLE

__HAL_RCC_UART4_CLK_ENABLE

__HAL_RCC_UART5_CLK_ENABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_I2C2_CLK_ENABLE

__HAL_RCC_I2C3_CLK_ENABLE

__HAL_RCC_I2C4_CLK_ENABLE

__HAL_RCC_CRS_CLK_ENABLE

__HAL_RCC_CAN1_CLK_ENABLE

__HAL_RCC_PWR_CLK_ENABLE

__HAL_RCC_DAC1_CLK_ENABLE

__HAL_RCC_OPAMP_CLK_ENABLE

__HAL_RCC_LPTIM1_CLK_ENABLE

__HAL_RCC_LPUART1_CLK_ENABLE

__HAL_RCC_LPTIM2_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_TIM3_CLK_DISABLE

__HAL_RCC_TIM4_CLK_DISABLE

__HAL_RCC_TIM5_CLK_DISABLE

__HAL_RCC_TIM6_CLK_DISABLE

__HAL_RCC_TIM7_CLK_DISABLE

__HAL_RCC_RTCAPB_CLK_DISABLE

__HAL_RCC_SPI2_CLK_DISABLE

__HAL_RCC_SPI3_CLK_DISABLE

__HAL_RCC_USART2_CLK_DISABLE

__HAL_RCC_USART3_CLK_DISABLE

__HAL_RCC_UART4_CLK_DISABLE

__HAL_RCC_UART5_CLK_DISABLE

__HAL_RCC_I2C1_CLK_DISABLE

__HAL_RCC_I2C2_CLK_DISABLE

__HAL_RCC_I2C3_CLK_DISABLE

__HAL_RCC_I2C4_CLK_DISABLE

__HAL_RCC_CRS_CLK_DISABLE

__HAL_RCC_CAN1_CLK_DISABLE

__HAL_RCC_PWR_CLK_DISABLE

__HAL_RCC_DAC1_CLK_DISABLE

__HAL_RCC_OPAMP_CLK_DISABLE

__HAL_RCC_LPTIM1_CLK_DISABLE

__HAL_RCC_LPUART1_CLK_DISABLE

__HAL_RCC_LPTIM2_CLK_DISABLE

APB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_TIM3_IS_CLK_ENABLED

__HAL_RCC_TIM4_IS_CLK_ENABLED

__HAL_RCC_TIM5_IS_CLK_ENABLED

__HAL_RCC_TIM6_IS_CLK_ENABLED

__HAL_RCC_TIM7_IS_CLK_ENABLED

__HAL_RCC_RTCAPB_IS_CLK_ENABLED

__HAL_RCC_WWDG_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_SPI3_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART3_IS_CLK_ENABLED

__HAL_RCC_UART4_IS_CLK_ENABLED

__HAL_RCC_UART5_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_I2C3_IS_CLK_ENABLED

__HAL_RCC_I2C4_IS_CLK_ENABLED

__HAL_RCC_CRS_IS_CLK_ENABLED

__HAL_RCC_CAN1_IS_CLK_ENABLED

__HAL_RCC_PWR_IS_CLK_ENABLED

__HAL_RCC_DAC1_IS_CLK_ENABLED

__HAL_RCC_OPAMP_IS_CLK_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_ENABLED

__HAL_RCC_LPUART1_IS_CLK_ENABLED

__HAL_RCC_LPTIM2_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM3_IS_CLK_DISABLED

__HAL_RCC_TIM4_IS_CLK_DISABLED

__HAL_RCC_TIM5_IS_CLK_DISABLED

__HAL_RCC_TIM6_IS_CLK_DISABLED

__HAL_RCC_TIM7_IS_CLK_DISABLED

__HAL_RCC_RTCAPB_IS_CLK_DISABLED

__HAL_RCC_WWDG_IS_CLK_DISABLED

__HAL_RCC_SPI2_IS_CLK_DISABLED

__HAL_RCC_SPI3_IS_CLK_DISABLED

__HAL_RCC_USART2_IS_CLK_DISABLED

__HAL_RCC_USART3_IS_CLK_DISABLED

__HAL_RCC_UART4_IS_CLK_DISABLED

__HAL_RCC_UART5_IS_CLK_DISABLED

__HAL_RCC_I2C1_IS_CLK_DISABLED

__HAL_RCC_I2C2_IS_CLK_DISABLED

__HAL_RCC_I2C3_IS_CLK_DISABLED

__HAL_RCC_I2C4_IS_CLK_DISABLED

__HAL_RCC_CRS_IS_CLK_DISABLED

__HAL_RCC_CAN1_IS_CLK_DISABLED

__HAL_RCC_PWR_IS_CLK_DISABLED

__HAL_RCC_DAC1_IS_CLK_DISABLED

__HAL_RCC_OPAMP_IS_CLK_DISABLED

__HAL_RCC_LPTIM1_IS_CLK_DISABLED

__HAL_RCC_LPUART1_IS_CLK_DISABLED

__HAL_RCC_LPTIM2_IS_CLK_DISABLED

APB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM2_CLK_SLEEP_ENABLE

__HAL_RCC_TIM3_CLK_SLEEP_ENABLE

__HAL_RCC_TIM4_CLK_SLEEP_ENABLE

__HAL_RCC_TIM5_CLK_SLEEP_ENABLE

__HAL_RCC_TIM6_CLK_SLEEP_ENABLE

__HAL_RCC_TIM7_CLK_SLEEP_ENABLE

__HAL_RCC_RTCAPB_CLK_SLEEP_ENABLE

__HAL_RCC_WWDG_CLK_SLEEP_ENABLE

__HAL_RCC_SPI2_CLK_SLEEP_ENABLE

__HAL_RCC_SPI3_CLK_SLEEP_ENABLE

__HAL_RCC_USART2_CLK_SLEEP_ENABLE

__HAL_RCC_USART3_CLK_SLEEP_ENABLE

__HAL_RCC_UART4_CLK_SLEEP_ENABLE

__HAL_RCC_UART5_CLK_SLEEP_ENABLE

__HAL_RCC_I2C1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C2_CLK_SLEEP_ENABLE

__HAL_RCC_I2C3_CLK_SLEEP_ENABLE

__HAL_RCC_I2C4_CLK_SLEEP_ENABLE

__HAL_RCC_CRS_CLK_SLEEP_ENABLE

__HAL_RCC_CAN1_CLK_SLEEP_ENABLE

__HAL_RCC_PWR_CLK_SLEEP_ENABLE

__HAL_RCC_DAC1_CLK_SLEEP_ENABLE

__HAL_RCC_OPAMP_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_LPUART1_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM2_CLK_SLEEP_ENABLE

__HAL_RCC_TIM2_CLK_SLEEP_DISABLE

__HAL_RCC_TIM3_CLK_SLEEP_DISABLE

__HAL_RCC_TIM4_CLK_SLEEP_DISABLE

__HAL_RCC_TIM5_CLK_SLEEP_DISABLE

__HAL_RCC_TIM6_CLK_SLEEP_DISABLE

__HAL_RCC_TIM7_CLK_SLEEP_DISABLE

__HAL_RCC_RTCAPB_CLK_SLEEP_DISABLE

__HAL_RCC_WWDG_CLK_SLEEP_DISABLE

__HAL_RCC_SPI2_CLK_SLEEP_DISABLE

__HAL_RCC_SPI3_CLK_SLEEP_DISABLE

__HAL_RCC_USART2_CLK_SLEEP_DISABLE

__HAL_RCC_USART3_CLK_SLEEP_DISABLE

__HAL_RCC_UART4_CLK_SLEEP_DISABLE

__HAL_RCC_UART5_CLK_SLEEP_DISABLE

__HAL_RCC_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C2_CLK_SLEEP_DISABLE

__HAL_RCC_I2C3_CLK_SLEEP_DISABLE

__HAL_RCC_I2C4_CLK_SLEEP_DISABLE

__HAL_RCC_CRS_CLK_SLEEP_DISABLE

__HAL_RCC_CAN1_CLK_SLEEP_DISABLE

__HAL_RCC_PWR_CLK_SLEEP_DISABLE

__HAL_RCC_DAC1_CLK_SLEEP_DISABLE

__HAL_RCC_OPAMP_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_LPUART1_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM2_CLK_SLEEP_DISABLE

APB1 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_TIM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM6_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM7_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RTCAPB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_WWDG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CRS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CAN1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_PWR_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DAC1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_OPAMP_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPUART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM5_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM6_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM7_IS_CLK_SLEEP_DISABLED

__HAL_RCC_RTCAPB_IS_CLK_SLEEP_DISABLED

__HAL_RCC_WWDG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_UART4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_UART5_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CRS_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CAN1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_PWR_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DAC1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_OPAMP_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPTIM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPUART1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPTIM2_IS_CLK_SLEEP_DISABLED

APB1 Peripheral Force Release Reset

__HAL_RCC_APB1_FORCE_RESET

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_TIM3_FORCE_RESET

__HAL_RCC_TIM4_FORCE_RESET

__HAL_RCC_TIM5_FORCE_RESET

__HAL_RCC_TIM6_FORCE_RESET

__HAL_RCC_TIM7_FORCE_RESET

__HAL_RCC_SPI2_FORCE_RESET

__HAL_RCC_SPI3_FORCE_RESET

__HAL_RCC_USART2_FORCE_RESET

__HAL_RCC_USART3_FORCE_RESET

__HAL_RCC_UART4_FORCE_RESET

__HAL_RCC_UART5_FORCE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_I2C2_FORCE_RESET

__HAL_RCC_I2C3_FORCE_RESET

__HAL_RCC_I2C4_FORCE_RESET

__HAL_RCC_CRS_FORCE_RESET

__HAL_RCC_CAN1_FORCE_RESET

__HAL_RCC_PWR_FORCE_RESET

__HAL_RCC_DAC1_FORCE_RESET

__HAL_RCC_OPAMP_FORCE_RESET

__HAL_RCC_LPTIM1_FORCE_RESET

__HAL_RCC_LPUART1_FORCE_RESET

__HAL_RCC_LPTIM2_FORCE_RESET

__HAL_RCC_APB1_RELEASE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_TIM3_RELEASE_RESET

__HAL_RCC_TIM4_RELEASE_RESET

__HAL_RCC_TIM5_RELEASE_RESET

__HAL_RCC_TIM6_RELEASE_RESET

__HAL_RCC_TIM7_RELEASE_RESET

__HAL_RCC_SPI2_RELEASE_RESET

__HAL_RCC_SPI3_RELEASE_RESET

__HAL_RCC_USART2_RELEASE_RESET

__HAL_RCC_USART3_RELEASE_RESET

__HAL_RCC_UART4_RELEASE_RESET

__HAL_RCC_UART5_RELEASE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

__HAL_RCC_I2C2_RELEASE_RESET

__HAL_RCC_I2C3_RELEASE_RESET

__HAL_RCC_I2C4_RELEASE_RESET

__HAL_RCC_CRS_RELEASE_RESET

__HAL_RCC_CAN1_RELEASE_RESET

__HAL_RCC_PWR_RELEASE_RESET

__HAL_RCC_DAC1_RELEASE_RESET

__HAL_RCC_OPAMP_RELEASE_RESET

__HAL_RCC_LPTIM1_RELEASE_RESET

__HAL_RCC_LPUART1_RELEASE_RESET

__HAL_RCC_LPTIM2_RELEASE_RESET

APB2 Peripheral Clock Enable Disable

__HAL_RCC_SYSCFG_CLK_ENABLE

__HAL_RCC_FIREWALL_CLK_ENABLE

__HAL_RCC_TIM1_CLK_ENABLE

__HAL_RCC_SPI1_CLK_ENABLE

__HAL_RCC_TIM8_CLK_ENABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_TIM15_CLK_ENABLE

__HAL_RCC_TIM16_CLK_ENABLE

__HAL_RCC_TIM17_CLK_ENABLE

__HAL_RCC_SAI1_CLK_ENABLE

__HAL_RCC_SAI2_CLK_ENABLE

__HAL_RCC_DFSDM1_CLK_ENABLE

__HAL_RCC_LTDC_CLK_ENABLE

__HAL_RCC_DSI_CLK_ENABLE

__HAL_RCC_SYSCFG_CLK_DISABLE

__HAL_RCC_TIM1_CLK_DISABLE

__HAL_RCC_SPI1_CLK_DISABLE

__HAL_RCC_TIM8_CLK_DISABLE

__HAL_RCC_USART1_CLK_DISABLE

__HAL_RCC_TIM15_CLK_DISABLE

__HAL_RCC_TIM16_CLK_DISABLE

__HAL_RCC_TIM17_CLK_DISABLE

__HAL_RCC_SAI1_CLK_DISABLE

__HAL_RCC_SAI2_CLK_DISABLE

__HAL_RCC_DFSDM1_CLK_DISABLE

__HAL_RCC_LTDC_CLK_DISABLE

__HAL_RCC_DSI_CLK_DISABLE

APB2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_SYSCFG_IS_CLK_ENABLED

__HAL_RCC_FIREWALL_IS_CLK_ENABLED

__HAL_RCC_TIM1_IS_CLK_ENABLED

__HAL_RCC_SPI1_IS_CLK_ENABLED

__HAL_RCC_TIM8_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_TIM15_IS_CLK_ENABLED

__HAL_RCC_TIM16_IS_CLK_ENABLED

__HAL_RCC_TIM17_IS_CLK_ENABLED

__HAL_RCC_SAI1_IS_CLK_ENABLED

__HAL_RCC_SAI2_IS_CLK_ENABLED

__HAL_RCC_DFSDM1_IS_CLK_ENABLED

__HAL_RCC_LTDC_IS_CLK_ENABLED

__HAL_RCC_DSI_IS_CLK_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_DISABLED

__HAL_RCC_TIM1_IS_CLK_DISABLED

__HAL_RCC_SPI1_IS_CLK_DISABLED

__HAL_RCC_TIM8_IS_CLK_DISABLED

__HAL_RCC_USART1_IS_CLK_DISABLED

__HAL_RCC_TIM15_IS_CLK_DISABLED

__HAL_RCC_TIM16_IS_CLK_DISABLED

__HAL_RCC_TIM17_IS_CLK_DISABLED

__HAL_RCC_SAI1_IS_CLK_DISABLED

__HAL_RCC_SAI2_IS_CLK_DISABLED

__HAL_RCC_DFSDM1_IS_CLK_DISABLED

__HAL_RCC_LTDC_IS_CLK_DISABLED

__HAL_RCC_DSI_IS_CLK_DISABLED

APB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE

__HAL_RCC_TIM1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM8_CLK_SLEEP_ENABLE

__HAL_RCC_USART1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM15_CLK_SLEEP_ENABLE

__HAL_RCC_TIM16_CLK_SLEEP_ENABLE

__HAL_RCC_TIM17_CLK_SLEEP_ENABLE

__HAL_RCC_SAI1_CLK_SLEEP_ENABLE

__HAL_RCC_SAI2_CLK_SLEEP_ENABLE

__HAL_RCC_DFSDM1_CLK_SLEEP_ENABLE

__HAL_RCC_LTDC_CLK_SLEEP_ENABLE

__HAL_RCC_DSI_CLK_SLEEP_ENABLE

__HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE

__HAL_RCC_TIM1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM8_CLK_SLEEP_DISABLE

__HAL_RCC_USART1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM15_CLK_SLEEP_DISABLE

__HAL_RCC_TIM16_CLK_SLEEP_DISABLE

__HAL_RCC_TIM17_CLK_SLEEP_DISABLE

__HAL_RCC_SAI1_CLK_SLEEP_DISABLE

__HAL_RCC_SAI2_CLK_SLEEP_DISABLE

__HAL_RCC_DFSDM1_CLK_SLEEP_DISABLE

__HAL_RCC_LTDC_CLK_SLEEP_DISABLE

__HAL_RCC_DSI_CLK_SLEEP_DISABLE

APB2 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM8_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM15_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM16_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM17_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SAI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SAI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DFSDM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LTDC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DSI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM8_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM15_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM16_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM17_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SAI1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SAI2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DFSDM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LTDC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DSI_IS_CLK_SLEEP_DISABLED

APB2 Peripheral Force Release Reset

__HAL_RCC_APB2_FORCE_RESET

__HAL_RCC_SYSCFG_FORCE_RESET

__HAL_RCC_TIM1_FORCE_RESET

__HAL_RCC_SPI1_FORCE_RESET

__HAL_RCC_TIM8_FORCE_RESET

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_TIM15_FORCE_RESET

__HAL_RCC_TIM16_FORCE_RESET

__HAL_RCC_TIM17_FORCE_RESET

__HAL_RCC_SAI1_FORCE_RESET

__HAL_RCC_SAI2_FORCE_RESET

__HAL_RCC_DFSDM1_FORCE_RESET

__HAL_RCC_LTDC_FORCE_RESET

__HAL_RCC_DSI_FORCE_RESET

__HAL_RCC_APB2_RELEASE_RESET

__HAL_RCC_SYSCFG_RELEASE_RESET

__HAL_RCC_TIM1_RELEASE_RESET

__HAL_RCC_SPI1_RELEASE_RESET

__HAL_RCC_TIM8_RELEASE_RESET

__HAL_RCC_USART1_RELEASE_RESET

__HAL_RCC_TIM15_RELEASE_RESET

__HAL_RCC_TIM16_RELEASE_RESET

__HAL_RCC_TIM17_RELEASE_RESET

__HAL_RCC_SAI1_RELEASE_RESET

__HAL_RCC_SAI2_RELEASE_RESET

__HAL_RCC_DFSDM1_RELEASE_RESET

__HAL_RCC_LTDC_RELEASE_RESET

__HAL_RCC_DSI_RELEASE_RESET

RCC Backup Domain Reset

__HAL_RCC_BACKUPRESET_FORCE

Description:

- Macros to force or release the Backup domain reset.

Return value:

- None

Notes:

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_CSR register. The BKPSRAM is not affected by this reset.

__HAL_RCC_BACKUPRESET_RELEASE

RCC Exported Macros

__HAL_RCC_HSI_ENABLE

Description:

- Macros to enable or disable the Internal High Speed 16MHz oscillator (HSI).

Return value:

- None

Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. This parameter can be: ENABLE or DISABLE. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_DISABLE

__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the Internal High Speed 16MHz oscillator (HSI) calibration value.

Parameters:

- __HSICALIBRATIONVALUE__: specifies the calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between 0 and 0x1F (STM32L43x/STM32L44x/STM32L47x/STM32L48x) or 0x7F (for other devices).

Return value:

- None

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

__HAL_RCC_HSIAUTOMATIC_START_ENABLE

Description:

- Macros to enable or disable the wakeup the Internal High Speed oscillator (HSI) in parallel to the Internal Multi Speed oscillator (MSI) used at system wakeup.

Return value:

- None

Notes:

- The enable of this function has not effect on the HSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSIAUTOMATIC_START_DISABLE

__HAL_RCC_HSISTOP_ENABLE

Description:

- Macros to enable or disable the force of the Internal High Speed oscillator (HSI) in STOP mode to be quickly available as kernel clock for USARTs and I2Cs.

Return value:

- None

Notes:

- Keeping the HSI ON in STOP mode allows to avoid slowing down the communication speed because of the HSI startup time. The enable of this function has not effect on the HSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSISTOP_DISABLE

__HAL_RCC_MSI_ENABLE

Description:

- Macros to enable or disable the Internal Multi Speed oscillator (MSI).

Return value:

- None

Notes:

- The MSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). MSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the MSI. After enabling the MSI, the application software should wait on MSIRDY flag to be set indicating that MSI clock is stable and can be used as system clock source. When the MSI is stopped, MSIRDY flag goes low after 6 MSI oscillator clock cycles.

__HAL_RCC_MSI_DISABLE

__HAL_RCC_MSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro Adjusts the Internal Multi Speed oscillator (MSI) calibration value.

Parameters:

- __MSICALIBRATIONVALUE__: specifies the calibration trimming value (default is RCC_MSICALIBRATION_DEFAULT). This parameter must be a number between 0 and 255.

Return value:

- None

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal MSI RC. Refer to the Application Note AN3300 for more details on how to calibrate the MSI.

__HAL_RCC_MSI_RANGE_CONFIG

Description:

- Macro configures the Internal Multi Speed oscillator (MSI) clock range in run mode.

Parameters:

- __MSIRANGEVALUE__: specifies the MSI clock range. This parameter must be one of the following values:
  - RCC_MSIRANGE_0 MSI clock is around 100 KHz
  - RCC_MSIRANGE_1 MSI clock is around 200 KHz
  - RCC_MSIRANGE_2 MSI clock is around 400 KHz
  - RCC_MSIRANGE_3 MSI clock is around 800 KHz
  - RCC_MSIRANGE_4 MSI clock is around 1 MHz
  - RCC_MSIRANGE_5 MSI clock is around 2 MHz
  - RCC_MSIRANGE_6 MSI clock is around 4 MHz (default after Reset)
  - RCC_MSIRANGE_7 MSI clock is around 8 MHz
  - RCC_MSIRANGE_8 MSI clock is around 16 MHz
  - RCC_MSIRANGE_9 MSI clock is around 24 MHz
  - RCC_MSIRANGE_10 MSI clock is around 32 MHz
  - RCC_MSIRANGE_11 MSI clock is around 48 MHz

Return value:

- None

Notes:

- After restart from Reset , the MSI clock is around 4 MHz. After stop the startup clock can be MSI (at any of its possible frequencies, the one that was used before entering stop mode) or HSI. After Standby its frequency can be selected between 4 possible values (1, 2, 4 or 8 MHz). MSIRANGE can be modified when MSI is OFF (MSION=0) or when MSI is ready (MSIRDY=1). The MSI clock range after reset can be modified on the fly.

__HAL_RCC_MSI_STANDBY_RANGE_CONFIG

Description:

- Macro configures the Internal Multi Speed oscillator (MSI) clock range after Standby mode After Standby its frequency can be selected between 4 possible values (1, 2, 4 or 8 MHz).

Parameters:

- __MSIRANGEVALUE__: specifies the MSI clock range. This parameter must be one

of the following values:
- – RCC_MSIRANGE_4 MSI clock is around 1 MHz
- – RCC_MSIRANGE_5 MSI clock is around 2 MHz
- – RCC_MSIRANGE_6 MSI clock is around 4 MHz (default after Reset)
- – RCC_MSIRANGE_7 MSI clock is around 8 MHz

Return value:

- None

__HAL_RCC_GET_MSI_RANGE

Description:

- Macro to get the Internal Multi Speed oscillator (MSI) clock range in run mode.

Return value:

- MSI: clock range. This parameter must be one of the following values:
  - – RCC_MSIRANGE_0 MSI clock is around 100 KHz
  - – RCC_MSIRANGE_1 MSI clock is around 200 KHz
  - – RCC_MSIRANGE_2 MSI clock is around 400 KHz
  - – RCC_MSIRANGE_3 MSI clock is around 800 KHz
  - – RCC_MSIRANGE_4 MSI clock is around 1 MHz
  - – RCC_MSIRANGE_5 MSI clock is around 2 MHz
  - – RCC_MSIRANGE_6 MSI clock is around 4 MHz (default after Reset)
  - – RCC_MSIRANGE_7 MSI clock is around 8 MHz
  - – RCC_MSIRANGE_8 MSI clock is around 16 MHz
  - – RCC_MSIRANGE_9 MSI clock is around 24 MHz
  - – RCC_MSIRANGE_10 MSI clock is around 32 MHz
  - – RCC_MSIRANGE_11 MSI clock is around 48 MHz

__HAL_RCC_LSI_ENABLE

Description:

- Macros to enable or disable the Internal Low Speed oscillator (LSI).

Return value:

- None

Notes:

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

__HAL_RCC_LSI_DISABLE

__HAL_RCC_HSE_CONFIG

Description:

- Macro to configure the External High Speed oscillator (HSE).

Parameters:

- __STATE__: specifies the new state of the HSE. This parameter can be one of the following values:
  - RCC_HSE_OFF Turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
  - RCC_HSE_ON Turn ON the HSE oscillator.
  - RCC_HSE_BYPASS HSE oscillator bypassed with external clock.

Return value:

- None

Notes:

- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass. After enabling the HSE (RCC_HSE_ON or RCC_HSE_Bypass), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY modes. This function reset the CSSON bit, so if the clock security system(CSS) was previously enabled you have to enable it again after calling this function.

__HAL_RCC_LSE_CONFIG

Description:

- Macro to configure the External Low

Speed oscillator (LSE).

Parameters:

- __STATE__: specifies the new state of the LSE. This parameter can be one of the following values:
  - RCC_LSE_OFF Turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
  - RCC_LSE_ON Turn ON the LSE oscillator.
  - RCC_LSE_BYPASS LSE oscillator bypassed with external clock.

Return value:

- None

Notes:

- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC_LSE_ON or RCC_LSE_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

__HAL_RCC_HSI48_ENABLE

Description:

- Macros to enable or disable the Internal High Speed 48MHz oscillator (HSI48).

Return value:

- None

Notes:

- The HSI48 is stopped by hardware when entering STOP and STANDBY modes. After enabling the HSI48, the application software should wait on HSI48RDY flag to be set indicating that HSI48 clock is stable. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSI48_DISABLE

__HAL_RCC_RTC_CONFIG

Description:

- Macros to configure the RTC clock

(RTCCLK).

Parameters:

- __RTC_CLKSOURCE__: specifies the RTC clock source. This parameter can be one of the following values:
  - RCC_RTCCLKSOURCE_NO_CLK No clock selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock.
  - RCC_RTCCLKSOURCE_HSE_DIV32 HSE clock divided by 32 selected

Return value:

- None

Notes:

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it cannot be changed unless the Backup domain is reset using __HAL_RCC_BACKUPRESET_FORCE() macro, or by a Power On Reset (POR).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

__HAL_RCC_GET_RTC_SOURCE

Description:

- Macro to get the RTC clock source.

Return value:

- The: returned value can be one of the following:
  - RCC_RTCCLKSOURCE_NO_CLK No clock selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock.
  - RCC_RTCCLKSOURCE_HSE_DIV32 HSE clock divided by 32 selected

__HAL_RCC_PLL_ENABLE

Description:

- Macros to enable or disable the main PLL.

Return value:

- None

Notes:

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL can not be disabled if it is used as system clock source The main PLL is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL_DISABLE

__HAL_RCC_PLL_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL clock source.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_NONE No clock selected as PLL clock entry
  - RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry

Return value:

- None

Notes:

- This function must be used only when the main PLL is disabled.
- This clock source is common for the main PLL and audio PLL (PLLSAI1 and PLLSAI2).

__HAL_RCC_PLL_PLLM_CONFIG

Description:

- Macro to configure the PLL source division factor M.

Parameters:

- __PLLM__: specifies the division factor for

PLL VCO input clock This parameter must be a number between Min_Data = 1 and Max_Data = 16 on STM32L4Rx/STM32L4Sx devices. This parameter must be a number between Min_Data = 1 and Max_Data = 8 on other devices.

Return value:

- None

Notes:

- This function must be used only when the main PLL is disabled.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 4 to 16 MHz. It is recommended to select a frequency of 16 MHz to limit PLL jitter.

__HAL_RCC_PLL_CONFIG

Description:

- Macro to configure the main PLL clock source, multiplication and division factors.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_NONE No clock selected as PLL clock entry
  - RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
- __PLLM__: specifies the division factor for PLL VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16 on STM32L4Rx/STM32L4Sx devices. This parameter must be a number between Min_Data = 1 and Max_Data = 8 on other devices.
- __PLLN__: specifies the multiplication factor for PLL VCO output clock. This parameter must be a number between 8 and 86.
- __PLLP__: specifies the division factor for SAI clock. This parameter must be a number in the range (7 or 17) for STM32L47x/STM32L48x else (2 to 31).

- __PLLQ__: specifies the division factor for OTG FS, SDMMC1 and RNG clocks. This parameter must be in the range (2, 4, 6 or 8).
- __PLLR__: specifies the division factor for the main system clock.

Return value:

- None

Notes:

- This function must be used only when the main PLL is disabled.
- This clock source is common for the main PLL and audio PLL (PLLSAI1 and PLLSAI2).
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 4 to 16 MHz. It is recommended to select a frequency of 16 MHz to limit PLL jitter.
- You have to set the PLLN parameter correctly to ensure that the VCO output frequency is between 64 and 344 MHz.
- If the USB OTG FS is used in your application, you have to set the PLLQ parameter correctly to have 48 MHz clock for the USB. However, the SDMMC1 and RNG need a frequency lower than or equal to 48 MHz to work correctly.
- You have to set the PLLR parameter correctly to not exceed 80MHZ. This parameter must be in the range (2, 4, 6 or 8).

__HAL_RCC_GET_PLL_OSCSOURCE

Description:

- Macro to get the oscillator used as PLL clock source.

Return value:

- The: oscillator used as PLL clock source. The returned value can be one of the following:
  - RCC_PLLSOURCE_NONE: No oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_MSI: MSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSI: HSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSE: HSE oscillator is used as PLL clock source.

__HAL_RCC_PLLCLKOUT_ENABLE

Description:

- Enable or disable each clock output

(RCC_PLL_SYSCLK, RCC_PLL_48M1CLK, RCC_PLL_SAI3CLK)

Parameters:

- __PLLCLOCKOUT__: specifies the PLL clock to be output. This parameter can be one or a combination of the following values:
  - RCC_PLL_SAI3CLK This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI interface in case.
  - RCC_PLL_48M1CLK This Clock is used to generate the clock for the USB OTG FS (48 MHz), the random analog generator (<=48 MHz) and the SDMMC1 (<= 48 MHz).
  - RCC_PLL_SYSCLK This Clock is used to generate the high speed system clock (up to 80MHz)

Return value:

- None

Notes:

- Enabling/disabling clock outputs RCC_PLL_SAI3CLK and RCC_PLL_48M1CLK can be done at anytime without the need to stop the PLL in order to save power. But RCC_PLL_SYSCLK cannot be stopped if used as System Clock.

__HAL_RCC_PLLCLKOUT_DISABLE

__HAL_RCC_GET_PLLCLKOUT_CONFIG

Description:

- Get clock output enable status (RCC_PLL_SYSCLK, RCC_PLL_48M1CLK, RCC_PLL_SAI3CLK)

Parameters:

- __PLLCLOCKOUT__: specifies the output PLL clock to be checked. This parameter can be one of the following values:
  - RCC_PLL_SAI3CLK This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI interface in case.
  - RCC_PLL_48M1CLK This Clock is used to generate the clock for the USB OTG FS (48 MHz), the random analog generator (<=48 MHz) and the SDMMC1 (<= 48 MHz).

- RCC_PLL_SYSCLK This Clock is used to generate the high speed system clock (up to 80MHz)

Return value:

- SET: / RESET

__HAL_RCC_SYSCLK_CONFIG

Description:

- Macro to configure the system clock source.

Parameters:

- __SYSCLKSOURCE__: specifies the system clock source. This parameter can be one of the following values:
  - RCC_SYSCLKSOURCE_MSI: MSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSI: HSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSE: HSE oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_PLLCLK: PLL output is used as system clock source.

Return value:

- None

__HAL_RCC_GET_SYSCLK_SOURCE

Description:

- Macro to get the clock source used as system clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
  - RCC_SYSCLKSOURCE_STATUS_MSI: MSI used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_HSI: HSI used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_HSE: HSE used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_PLLCLK: PLL used as system clock.

__HAL_RCC_LSEDRIVE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE) drive capability.

Parameters:

- __LSEDRIVE__: specifies the new state of

the LSE drive capability. This parameter can be one of the following values:
- RCC_LSEDRIVE_LOW LSE oscillator low drive capability.
- RCC_LSEDRIVE_MEDIUMLOW LSE oscillator medium low drive capability.
- RCC_LSEDRIVE_MEDIUMHIGH LSE oscillator medium high drive capability.
- RCC_LSEDRIVE_HIGH LSE oscillator high drive capability.

Return value:
- None

Notes:
- As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset).

__HAL_RCC_WAKEUPSTOP_CLK_CONFIG

Description:
- Macro to configure the wake up from stop clock.

Parameters:
- __STOPWUCLK__: specifies the clock source used after wake up from stop. This parameter can be one of the following values:
  - RCC_STOP_WAKEUPCLOCK_MSI MSI selected as system clock source
  - RCC_STOP_WAKEUPCLOCK_HSI HSI selected as system clock source

Return value:
- None

__HAL_RCC_MCO1_CONFIG

Description:
- Macro to configure the MCO clock.

Parameters:
- __MCOCLKSOURCE__: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_NOCLOCK MCO output disabled
  - RCC_MCO1SOURCE_SYSCLK System clock selected as MCO source
  - RCC_MCO1SOURCE_MSI MSI clock selected as MCO source

- RCC_MCO1SOURCE_HSI HSI clock selected as MCO source
- RCC_MCO1SOURCE_HSE HSE clock selected as MCO sourcee
- RCC_MCO1SOURCE_PLLCLK Main PLL clock selected as MCO source
- RCC_MCO1SOURCE_LSI LSI clock selected as MCO source
- RCC_MCO1SOURCE_LSE LSE clock selected as MCO source

- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 MCO clock source is divided by 1
  - RCC_MCODIV_2 MCO clock source is divided by 2
  - RCC_MCODIV_4 MCO clock source is divided by 4
  - RCC_MCODIV_8 MCO clock source is divided by 8
  - RCC_MCODIV_16 MCO clock source is divided by 16

Flags

RCC_FLAG_MSIRDY	MSI Ready flag
RCC_FLAG_HSIRDY	HSI Ready flag
RCC_FLAG_HSERDY	HSE Ready flag
RCC_FLAG_PLLRDY	PLL Ready flag
RCC_FLAG_PLLSAI1RDY	PLLSAI1 Ready flag
RCC_FLAG_PLLSAI2RDY	PLLSAI2 Ready flag
RCC_FLAG_LSERDY	LSE Ready flag
RCC_FLAG_LSECSSD	LSE Clock Security System Interrupt flag
RCC_FLAG_LSIRDY	LSI Ready flag
RCC_FLAG_RMVF	Remove reset flag
RCC_FLAG_FWRST	Firewall reset flag
RCC_FLAG_OBLRST	Option Byte Loader reset flag
RCC_FLAG_PINRST	PIN reset flag
RCC_FLAG_BORRST	BOR reset flag
RCC_FLAG_SFTRST	Software Reset flag
RCC_FLAG_IWDGRST	Independent Watchdog reset flag
RCC_FLAG_WWDGRST	Window watchdog reset flag
RCC_FLAG_LPWRRST	Low-Power reset flag
RCC_FLAG_HSI48RDY	HSI48 Ready flag

Flags Interrupts Management

__HAL_RCC_ENABLE_IT

Description:

- Enable RCC interrupt (Perform Byte access to RCC_CIR[14:8] bits to enable the selected interrupts).

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
  – RCC_IT_LSIRDY LSI ready interrupt
  – RCC_IT_LSERDY LSE ready interrupt
  – RCC_IT_MSIRDY HSI ready interrupt
  – RCC_IT_HSIRDY HSI ready interrupt
  – RCC_IT_HSERDY HSE ready interrupt
  – RCC_IT_PLLRDY Main PLL ready interrupt
  – RCC_IT_PLLSAI1RDY PLLSAI1 ready interrupt
  – RCC_IT_PLLSAI2RDY PLLSAI2 ready interrupt for devices with PLLSAI2
  – RCC_IT_LSECSS LSE Clock security system interrupt

Return value:

- None

__HAL_RCC_DISABLE_IT

Description:

- Disable RCC interrupt (Perform Byte access to RCC_CIR[14:8] bits to disable the selected interrupts).

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
  – RCC_IT_LSIRDY LSI ready interrupt
  – RCC_IT_LSERDY LSE ready interrupt
  – RCC_IT_MSIRDY HSI ready interrupt
  – RCC_IT_HSIRDY HSI ready interrupt
  – RCC_IT_HSERDY HSE ready interrupt
  – RCC_IT_PLLRDY Main PLL ready interrupt
  – RCC_IT_PLLSAI1RDY PLLSAI1 ready interrupt
  – RCC_IT_PLLSAI2RDY PLLSAI2 ready interrupt for devices with PLLSAI2
  – RCC_IT_LSECSS LSE Clock security system interrupt

Return value:

- None

__HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's interrupt pending bits (Perform Byte access to RCC_CIR[23:16] bits to clear the selected interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_MSIRDY MSI ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY Main PLL ready interrupt
  - RCC_IT_PLLSAI1RDY PLLSAI1 ready interrupt
  - RCC_IT_PLLSAI2RDY PLLSAI2 ready interrupt for devices with PLLSAI2
  - RCC_IT_CSS HSE Clock security system interrupt
  - RCC_IT_LSECSS LSE Clock security system interrupt

Return value:

- None

__HAL_RCC_GET_IT

Description:

- Check whether the RCC interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt source to check. This parameter can be one of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_MSIRDY MSI ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY Main PLL ready interrupt
  - RCC_IT_PLLSAI1RDY PLLSAI1 ready interrupt
  - RCC_IT_PLLSAI2RDY PLLSAI2 ready interrupt for devices with PLLSAI2
  - RCC_IT_CSS HSE Clock security system interrupt
  - RCC_IT_LSECSS LSE Clock security

system interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RCC_CLEAR_RESET_FLAGS

Description:

- Set RMVF bit to clear the reset flags.

Return value:

- None

__HAL_RCC_GET_FLAG

Description:

- Check whether the selected RCC flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_FLAG_MSIRDY MSI oscillator clock ready
  - RCC_FLAG_HSIRDY HSI oscillator clock ready
  - RCC_FLAG_HSERDY HSE oscillator clock ready
  - RCC_FLAG_PLLRDY Main PLL clock ready
  - RCC_FLAG_PLLSAI1RDY PLLSAI1 clock ready
  - RCC_FLAG_PLLSAI2RDY PLLSAI2 clock ready for devices with PLLSAI2
  - RCC_FLAG_LSERDY LSE oscillator clock ready
  - RCC_FLAG_LSECSSD Clock security system failure on LSE oscillator detection
  - RCC_FLAG_LSIRDY LSI oscillator clock ready
  - RCC_FLAG_BORRST BOR reset
  - RCC_FLAG_OBLRST OBLRST reset
  - RCC_FLAG_PINRST Pin reset
  - RCC_FLAG_FWRST FIREWALL reset
  - RCC_FLAG_RMVF Remove reset Flag
  - RCC_FLAG_SFTRST Software reset
  - RCC_FLAG_IWDGRST Independent Watchdog reset
  - RCC_FLAG_WWDGRST Window Watchdog reset
  - RCC_FLAG_LPWRRST Low Power reset

Return value:

- The: new state of __FLAG__ (TRUE or

FALSE).

HSE Config

RCC_HSE_OFF	HSE clock deactivation
RCC_HSE_ON	HSE clock activation
RCC_HSE_BYPASS	External clock source for HSE clock

HSI48 Config

RCC_HSI48_OFF	HSI48 clock deactivation
RCC_HSI48_ON	HSI48 clock activation

HSI Config

RCC_HSI_OFF	HSI clock deactivation
RCC_HSI_ON	HSI clock activation
RCC_HSICALIBRATION_DEFAULT	

Interrupts

RCC_IT_LSIRDY	LSI Ready Interrupt flag
RCC_IT_LSERDY	LSE Ready Interrupt flag
RCC_IT_MSIRDY	MSI Ready Interrupt flag
RCC_IT_HSIRDY	HSI16 Ready Interrupt flag
RCC_IT_HSERDY	HSE Ready Interrupt flag
RCC_IT_PLLRDY	PLL Ready Interrupt flag
RCC_IT_PLLSAI1RDY	PLLSAI1 Ready Interrupt flag
RCC_IT_PLLSAI2RDY	PLLSAI2 Ready Interrupt flag
RCC_IT_CSS	Clock Security System Interrupt flag
RCC_IT_LSECSS	LSE Clock Security System Interrupt flag
RCC_IT_HSI48RDY	HSI48 Ready Interrupt flag

LSE Drive Config

RCC_LSEDRIVE_LOW	LSE low drive capability
RCC_LSEDRIVE_MEDIUMLOW	LSE medium low drive capability
RCC_LSEDRIVE_MEDIUMHIGH	LSE medium high drive capability
RCC_LSEDRIVE_HIGH	LSE high drive capability

LSE Config

RCC_LSE_OFF	LSE clock deactivation
RCC_LSE_ON	LSE clock activation
RCC_LSE_BYPASS	External clock source for LSE clock

LSI Config

RCC_LSI_OFF	LSI clock deactivation
RCC_LSI_ON	LSI clock activation

MCO1 Clock Source

RCC_MCO1SOURCE_NOCLOCK	MCO1 output disabled, no clock on MCO1
RCC_MCO1SOURCE_SYSCLK	SYSCLK selection as MCO1 source
RCC_MCO1SOURCE_MSI	MSI selection as MCO1 source
RCC_MCO1SOURCE_HSI	HSI selection as MCO1 source
RCC_MCO1SOURCE_HSE	HSE selection as MCO1 source
RCC_MCO1SOURCE_PLLCLK	PLLCLK selection as MCO1 source
RCC_MCO1SOURCE_LSI	LSI selection as MCO1 source
RCC_MCO1SOURCE_LSE	LSE selection as MCO1 source
RCC_MCO1SOURCE_HSI48	HSI48 selection as MCO1 source (STM32L43x/STM32L44x devices)

MCO1 Clock Prescaler

RCC_MCODIV_1	MCO not divided
RCC_MCODIV_2	MCO divided by 2
RCC_MCODIV_4	MCO divided by 4
RCC_MCODIV_8	MCO divided by 8
RCC_MCODIV_16	MCO divided by 16

MCO Index

RCC_MCO1	
RCC_MCO	MCO1 to be compliant with other families with 2 MCOs

MSI Clock Range

RCC_MSIRANGE_0	MSI = 100 KHz
RCC_MSIRANGE_1	MSI = 200 KHz
RCC_MSIRANGE_2	MSI = 400 KHz
RCC_MSIRANGE_3	MSI = 800 KHz
RCC_MSIRANGE_4	MSI = 1 MHz
RCC_MSIRANGE_5	MSI = 2 MHz
RCC_MSIRANGE_6	MSI = 4 MHz
RCC_MSIRANGE_7	MSI = 8 MHz
RCC_MSIRANGE_8	MSI = 16 MHz
RCC_MSIRANGE_9	MSI = 24 MHz
RCC_MSIRANGE_10	MSI = 32 MHz
RCC_MSIRANGE_11	MSI = 48 MHz

MSI Config

RCC_MSI_OFF	MSI clock deactivation
RCC_MSI_ON	MSI clock activation
RCC_MSICALIBRATION_DEFAULT	Default MSI calibration trimming value

Oscillator Type

RCC_OSCILLATORTYPE_NONE	Oscillator configuration unchanged
RCC_OSCILLATORTYPE_HSE	HSE to configure
RCC_OSCILLATORTYPE_HSI	HSI to configure
RCC_OSCILLATORTYPE_LSE	LSE to configure
RCC_OSCILLATORTYPE_LSI	LSI to configure
RCC_OSCILLATORTYPE_MSI	MSI to configure
RCC_OSCILLATORTYPE_HSI48	HSI48 to configure

PLLP Clock Divider

RCC_PLLP_DIV2	PLLP division factor = 2
RCC_PLLP_DIV3	PLLP division factor = 3
RCC_PLLP_DIV4	PLLP division factor = 4
RCC_PLLP_DIV5	PLLP division factor = 5
RCC_PLLP_DIV6	PLLP division factor = 6
RCC_PLLP_DIV7	PLLP division factor = 7
RCC_PLLP_DIV8	PLLP division factor = 8
RCC_PLLP_DIV9	PLLP division factor = 9
RCC_PLLP_DIV10	PLLP division factor = 10
RCC_PLLP_DIV11	PLLP division factor = 11
RCC_PLLP_DIV12	PLLP division factor = 12
RCC_PLLP_DIV13	PLLP division factor = 13
RCC_PLLP_DIV14	PLLP division factor = 14
RCC_PLLP_DIV15	PLLP division factor = 15
RCC_PLLP_DIV16	PLLP division factor = 16
RCC_PLLP_DIV17	PLLP division factor = 17
RCC_PLLP_DIV18	PLLP division factor = 18
RCC_PLLP_DIV19	PLLP division factor = 19
RCC_PLLP_DIV20	PLLP division factor = 20
RCC_PLLP_DIV21	PLLP division factor = 21
RCC_PLLP_DIV22	PLLP division factor = 22
RCC_PLLP_DIV23	PLLP division factor = 23
RCC_PLLP_DIV24	PLLP division factor = 24
RCC_PLLP_DIV25	PLLP division factor = 25
RCC_PLLP_DIV26	PLLP division factor = 26
RCC_PLLP_DIV27	PLLP division factor = 27
RCC_PLLP_DIV28	PLLP division factor = 28

RCC_PLLP_DIV29	PLLP division factor = 29
RCC_PLLP_DIV30	PLLP division factor = 30
RCC_PLLP_DIV31	PLLP division factor = 31

PLLQ Clock Divider

RCC_PLLQ_DIV2	PLLQ division factor = 2
RCC_PLLQ_DIV4	PLLQ division factor = 4
RCC_PLLQ_DIV6	PLLQ division factor = 6
RCC_PLLQ_DIV8	PLLQ division factor = 8

PLLR Clock Divider

RCC_PLLR_DIV2	PLLR division factor = 2
RCC_PLLR_DIV4	PLLR division factor = 4
RCC_PLLR_DIV6	PLLR division factor = 6
RCC_PLLR_DIV8	PLLR division factor = 8

PLLSAI1 Clock Output

RCC_PLLSAI1_SAI1CLK	PLLSAI1CLK selection from PLLSAI1
RCC_PLLSAI1_48M2CLK	PLL48M2CLK selection from PLLSAI1
RCC_PLLSAI1_ADC1CLK	PLLADC1CLK selection from PLLSAI1

PLLSAI2 Clock Output

RCC_PLLSAI2_SAI2CLK	PLLSAI2CLK selection from PLLSAI2
RCC_PLLSAI2_DSICLK	PLLDSICLK selection from PLLSAI2
RCC_PLLSAI2_LTDCCLK	PLLLTDCCLK selection from PLLSAI2

PLL Clock Output

RCC_PLL_SAI3CLK	PLLSAI3CLK selection from main PLL (for devices with PLLSAI2)
RCC_PLL_48M1CLK	PLL48M1CLK selection from main PLL
RCC_PLL_SYSCLK	PLLCLK selection from main PLL

PLL Clock Source

RCC_PLLSOURCE_NONE	No clock selected as PLL entry clock source
RCC_PLLSOURCE_MSI	MSI clock selected as PLL entry clock source
RCC_PLLSOURCE_HSI	HSI clock selected as PLL entry clock source
RCC_PLLSOURCE_HSE	HSE clock selected as PLL entry clock source

PLL Config

RCC_PLL_NONE	PLL configuration unchanged
RCC_PLL_OFF	PLL deactivation
RCC_PLL_ON	PLL activation

RCC RTC Clock Configuration

__HAL_RCC_RTC_ENABLE	**Description:**

- Macros to enable or disable the RTC clock.

Return value:

- None

Notes:

- As the RTC is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the RTC (to be done once after reset). These macros must be used after the RTC clock source was selected.

__HAL_RCC_RTC_DISABLE

RTC Clock Source

RCC_RTCCLKSOURCE_NO_CLK	No clock used as RTC clock
RCC_RTCCLKSOURCE_LSE	LSE oscillator clock used as RTC clock
RCC_RTCCLKSOURCE_LSI	LSI oscillator clock used as RTC clock
RCC_RTCCLKSOURCE_HSE_DIV32	HSE oscillator clock divided by 32 used as RTC clock

Wake-Up from STOP Clock

RCC_STOP_WAKEUPCLOCK_MSI	MSI selection after wake-up from STOP
RCC_STOP_WAKEUPCLOCK_HSI	HSI selection after wake-up from STOP

System Clock Source

RCC_SYSCLKSOURCE_MSI	MSI selection as system clock
RCC_SYSCLKSOURCE_HSI	HSI selection as system clock
RCC_SYSCLKSOURCE_HSE	HSE selection as system clock
RCC_SYSCLKSOURCE_PLLCLK	PLL selection as system clock

System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_MSI	MSI used as system clock
RCC_SYSCLKSOURCE_STATUS_HSI	HSI used as system clock
RCC_SYSCLKSOURCE_STATUS_HSE	HSE used as system clock
RCC_SYSCLKSOURCE_STATUS_PLLCLK	PLL used as system clock

System Clock Type

RCC_CLOCKTYPE_SYSCLK	SYSCLK to configure
RCC_CLOCKTYPE_HCLK	HCLK to configure
RCC_CLOCKTYPE_PCLK1	PCLK1 to configure
RCC_CLOCKTYPE_PCLK2	PCLK2 to configure

Timeout Values

RCC_DBP_TIMEOUT_VALUE

RCC_LSE_TIMEOUT_VALUE

52 HAL RCC Extension Driver

52.1 RCCEx Firmware driver registers structures

52.1.1 RCC_PLLSAI1InitTypeDef

Data Fields

- ***uint32_t PLLSAI1Source***
- ***uint32_t PLLSAI1M***
- ***uint32_t PLLSAI1N***
- ***uint32_t PLLSAI1P***
- ***uint32_t PLLSAI1Q***
- ***uint32_t PLLSAI1R***
- ***uint32_t PLLSAI1ClockOut***

Field Documentation

- ***uint32_t RCC_PLLSAI1InitTypeDef::PLLSAI1Source***
  PLLSAI1Source: PLLSAI1 entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLLSAI1InitTypeDef::PLLSAI1M***
  PLLSAI1M: specifies the division factor for PLLSAI1 input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16
- ***uint32_t RCC_PLLSAI1InitTypeDef::PLLSAI1N***
  PLLSAI1N: specifies the multiplication factor for PLLSAI1 VCO output clock. This parameter must be a number between 8 and 86 or 127 depending on devices.
- ***uint32_t RCC_PLLSAI1InitTypeDef::PLLSAI1P***
  PLLSAI1P: specifies the division factor for SAI clock. This parameter must be a value of ***RCC_PLLP_Clock_Divider***
- ***uint32_t RCC_PLLSAI1InitTypeDef::PLLSAI1Q***
  PLLSAI1Q: specifies the division factor for USB/RNG/SDMMC1 clock. This parameter must be a value of ***RCC_PLLQ_Clock_Divider***
- ***uint32_t RCC_PLLSAI1InitTypeDef::PLLSAI1R***
  PLLSAI1R: specifies the division factor for ADC clock. This parameter must be a value of ***RCC_PLLR_Clock_Divider***
- ***uint32_t RCC_PLLSAI1InitTypeDef::PLLSAI1ClockOut***
  PLLSAIClockOut: specifies PLLSAI1 output clock to be enabled. This parameter must be a value of ***RCC_PLLSAI1_Clock_Output***

52.1.2 RCC_PLLSAI2InitTypeDef

Data Fields

- ***uint32_t PLLSAI2Source***
- ***uint32_t PLLSAI2M***
- ***uint32_t PLLSAI2N***
- ***uint32_t PLLSAI2P***
- ***uint32_t PLLSAI2Q***
- ***uint32_t PLLSAI2R***
- ***uint32_t PLLSAI2ClockOut***

Field Documentation

- ***uint32_t RCC_PLLSAI2InitTypeDef::PLLSAI2Source***
  PLLSAI2Source: PLLSAI2 entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLLSAI2InitTypeDef::PLLSAI2M***
  PLLSAI2M: specifies the division factor for PLLSAI2 input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16
- ***uint32_t RCC_PLLSAI2InitTypeDef::PLLSAI2N***
  PLLSAI2N: specifies the multiplication factor for PLLSAI2 VCO output clock. This parameter must be a number between 8 and 86 or 127 depending on devices.
- ***uint32_t RCC_PLLSAI2InitTypeDef::PLLSAI2P***
  PLLSAI2P: specifies the division factor for SAI clock. This parameter must be a value of ***RCC_PLLP_Clock_Divider***
- ***uint32_t RCC_PLLSAI2InitTypeDef::PLLSAI2Q***
  PLLSAI2Q: specifies the division factor for DSI clock. This parameter must be a value of ***RCC_PLLQ_Clock_Divider***
- ***uint32_t RCC_PLLSAI2InitTypeDef::PLLSAI2R***
  PLLSAI2R: specifies the division factor for ADC clock. This parameter must be a value of ***RCC_PLLR_Clock_Divider***
- ***uint32_t RCC_PLLSAI2InitTypeDef::PLLSAI2ClockOut***
  PLLSAIClockOut: specifies PLLSAI2 output clock to be enabled. This parameter must be a value of ***RCC_PLLSAI2_Clock_Output***

52.1.3 RCC_PeriphCLKInitTypeDef

Data Fields

- ***uint32_t PeriphClockSelection***
- ***RCC_PLLSAI1InitTypeDef PLLSAI1***
- ***RCC_PLLSAI2InitTypeDef PLLSAI2***
- ***uint32_t Usart1ClockSelection***
- ***uint32_t Usart2ClockSelection***
- ***uint32_t Usart3ClockSelection***
- ***uint32_t Uart4ClockSelection***
- ***uint32_t Uart5ClockSelection***
- ***uint32_t Lpuart1ClockSelection***
- ***uint32_t I2c1ClockSelection***
- ***uint32_t I2c2ClockSelection***
- ***uint32_t I2c3ClockSelection***
- ***uint32_t I2c4ClockSelection***
- ***uint32_t Lptim1ClockSelection***
- ***uint32_t Lptim2ClockSelection***
- ***uint32_t Sai1ClockSelection***
- ***uint32_t Sai2ClockSelection***
- ***uint32_t UsbClockSelection***
- ***uint32_t Sdmmc1ClockSelection***
- ***uint32_t RngClockSelection***
- ***uint32_t AdcClockSelection***
- ***uint32_t Dfsdm1ClockSelection***
- ***uint32_t Dfsdm1AudioClockSelection***
- ***uint32_t LtdcClockSelection***
- ***uint32_t DsiClockSelection***
- ***uint32_t OspiClockSelection***
- ***uint32_t RTCClockSelection***

Field Documentation

- ***uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection***
The Extended Clock to be configured. This parameter can be a value of ***RCCEx_Periph_Clock_Selection***
- ***RCC_PLLSAI1InitTypeDef RCC_PeriphCLKInitTypeDef::PLLSAI1***
PLLSAI1 structure parameters. This parameter will be used only when PLLSAI1 is selected as Clock Source for SAI1, USB/RNG/SDMMC1 or ADC
- ***RCC_PLLSAI2InitTypeDef RCC_PeriphCLKInitTypeDef::PLLSAI2***
PLLSAI2 structure parameters. This parameter will be used only when PLLSAI2 is selected as Clock Source for SAI2 or ADC
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection***
Specifies USART1 clock source. This parameter can be a value of ***RCCEx_USART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart2ClockSelection***
Specifies USART2 clock source. This parameter can be a value of ***RCCEx_USART2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart3ClockSelection***
Specifies USART3 clock source. This parameter can be a value of ***RCCEx_USART3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart4ClockSelection***
Specifies UART4 clock source. This parameter can be a value of ***RCCEx_UART4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart5ClockSelection***
Specifies UART5 clock source. This parameter can be a value of ***RCCEx_UART5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lpuart1ClockSelection***
Specifies LPUART1 clock source. This parameter can be a value of ***RCCEx_LPUART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection***
Specifies I2C1 clock source. This parameter can be a value of ***RCCEx_I2C1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c2ClockSelection***
Specifies I2C2 clock source. This parameter can be a value of ***RCCEx_I2C2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c3ClockSelection***
Specifies I2C3 clock source. This parameter can be a value of ***RCCEx_I2C3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c4ClockSelection***
Specifies I2C4 clock source. This parameter can be a value of ***RCCEx_I2C4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim1ClockSelection***
Specifies LPTIM1 clock source. This parameter can be a value of ***RCCEx_LPTIM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim2ClockSelection***
Specifies LPTIM2 clock source. This parameter can be a value of ***RCCEx_LPTIM2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai1ClockSelection***
Specifies SAI1 clock source. This parameter can be a value of ***RCCEx_SAI1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai2ClockSelection***
Specifies SAI2 clock source. This parameter can be a value of ***RCCEx_SAI2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection***
Specifies USB clock source (warning: same source for SDMMC1 and RNG). This parameter can be a value of ***RCCEx_USB_Clock_Source***

- ***uint32_t RCC_PeriphCLKInitTypeDef::Sdmmc1ClockSelection***
  Specifies SDMMC1 clock source (warning: same source for USB and RNG). This parameter can be a value of ***RCCEx_SDMMC1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RngClockSelection***
  Specifies RNG clock source (warning: same source for USB and SDMMC1). This parameter can be a value of ***RCCEx_RNG_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::AdcClockSelection***
  Specifies ADC interface clock source. This parameter can be a value of ***RCCEx_ADC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Dfsdm1ClockSelection***
  Specifies DFSDM1 clock source. This parameter can be a value of ***RCCEx_DFSDM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Dfsdm1AudioClockSelection***
  Specifies DFSDM1 audio clock source. This parameter can be a value of ***RCCEx_DFSDM1_Audio_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::LtdcClockSelection***
  Specifies LTDC clock source. This parameter can be a value of ***RCCEx_LTDC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::DsiClockSelection***
  Specifies DSI clock source. This parameter can be a value of ***RCCEx_DSI_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::OspiClockSelection***
  Specifies OctoSPI clock source. This parameter can be a value of ***RCCEx_OSPI_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection***
  Specifies RTC clock source. This parameter can be a value of ***RCC_RTC_Clock_Source***

52.1.4 RCC_CRSInitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t ReloadValue***
- ***uint32_t ErrorLimitValue***
- ***uint32_t HSI48CalibrationValue***

Field Documentation

- ***uint32_t RCC_CRSInitTypeDef::Prescaler***
  Specifies the division factor of the SYNC signal. This parameter can be a value of ***RCCEx_CRS_SynchroDivider***
- ***uint32_t RCC_CRSInitTypeDef::Source***
  Specifies the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroSource***
- ***uint32_t RCC_CRSInitTypeDef::Polarity***
  Specifies the input polarity for the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroPolarity***
- ***uint32_t RCC_CRSInitTypeDef::ReloadValue***
  Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro **__HAL_RCC_CRS_RELOADVALUE_CALCULATE(__FTARGET__, __FSYNC__)** This parameter must be a number between 0 and 0xFFFF or a value of ***RCCEx_CRS_ReloadValueDefault*** .

- ***uint32_t RCC_CRSInitTypeDef::ErrorLimitValue***
  Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of ***RCCEx_CRS_ErrorLimitDefault***
- ***uint32_t RCC_CRSInitTypeDef::HSI48CalibrationValue***
  Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x3F or a value of ***RCCEx_CRS_HSI48CalibrationDefault***

52.1.5 RCC_CRSSynchroInfoTypeDef

Data Fields

- ***uint32_t ReloadValue***
- ***uint32_t HSI48CalibrationValue***
- ***uint32_t FreqErrorCapture***
- ***uint32_t FreqErrorDirection***

Field Documentation

- ***uint32_t RCC_CRSSynchroInfoTypeDef::ReloadValue***
  Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::HSI48CalibrationValue***
  Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x3F
- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorCapture***
  Specifies the value loaded in the .FECAP, the frequency error counter value latched in the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorDirection***
  Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of ***RCCEx_CRS_FreqErrorDirection***

52.2 RCCEx Firmware driver API description

52.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

Important note: Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values.

This section contains the following APIs:

- ***HAL_RCCEx_PeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKFreq()***

52.2.2 Extended clock management functions

This subsection provides a set of functions allowing to control the activation or deactivation of MSI PLL-mode, PLLSAI1, PLLSAI2, LSE CSS, Low speed clock output and clock after wake-up from STOP mode.

This section contains the following APIs:

- *HAL_RCCEx_EnablePLLSAI1()*
- *HAL_RCCEx_DisablePLLSAI1()*
- *HAL_RCCEx_EnablePLLSAI2()*
- *HAL_RCCEx_DisablePLLSAI2()*
- *HAL_RCCEx_WakeUpStopCLKConfig()*
- *HAL_RCCEx_StandbyMSIRangeConfig()*
- *HAL_RCCEx_EnableLSECSS()*
- *HAL_RCCEx_DisableLSECSS()*
- *HAL_RCCEx_EnableLSECSS_IT()*
- *HAL_RCCEx_LSECSS_IRQHandler()*
- *HAL_RCCEx_LSECSS_Callback()*
- *HAL_RCCEx_EnableLSCO()*
- *HAL_RCCEx_DisableLSCO()*
- *HAL_RCCEx_EnableMSIPLLMode()*
- *HAL_RCCEx_DisableMSIPLLMode()*

52.2.3 Extended Clock Recovery System Control functions

For devices with Clock Recovery System feature (CRS), RCC Extention HAL driver can be used as follows:

1. In System clock config, HSI48 needs to be enabled
2. Enable CRS clock in IP MSP init which will use CRS functions
3. Call CRS functions as follows:
   a. Prepare synchronization configuration necessary for HSI48 calibration
      – Default values can be set for frequency Error Measurement (reload and error limit) and also HSI48 oscillator smooth trimming.
      – Macro __HAL_RCC_CRS_RELOADVALUE_CALCULATE can be also used to calculate directly reload value with target and sychronization frequencies values
   b. Call function HAL_RCCEx_CRSConfig which
      – Resets CRS registers to their default values.
      – Configures CRS registers with synchronization configuration
      – Enables automatic calibration and frequency error counter feature Note: When using USB LPM (Link Power Management) and the device is in Sleep mode, the periodic USB SOF will not be generated by the host. No SYNC signal will therefore be provided to the CRS to calibrate the HSI48 on the run. To guarantee the required clock precision after waking up from Sleep mode, the LSE or reference clock on the GPIOs should be used as SYNC signal.
   c. A polling function is provided to wait for complete synchronization
      – Call function HAL_RCCEx_CRSWaitSynchronization()
      – According to CRS status, user can decide to adjust again the calibration or continue application if synchronization is OK
4. User can retrieve information related to synchronization in calling function HAL_RCCEx_CRSGetSynchronizationInfo()
5. Regarding synchronization status and synchronization information, user can try a new calibration in changing synchronization configuration and call again

HAL_RCCEx_CRSConfig. Note: When the SYNC event is detected during the downcounting phase (before reaching the zero value), it means that the actual frequency is lower than the target (and so, that the TRIM value should be incremented), while when it is detected during the upcounting phase it means that the actual frequency is higher (and that the TRIM value should be decremented).

6. In interrupt mode, user can resort to the available macros (__HAL_RCC_CRS_XXX_IT). Interrupts will go through CRS Handler (CRS_IRQn/CRS_IRQHandler)
   - Call function HAL_RCCEx_CRSConfig()
   - Enable CRS_IRQn (thanks to NVIC functions)
   - Enable CRS interrupt (__HAL_RCC_CRS_ENABLE_IT)
   - Implement CRS status management in the following user callbacks called from HAL_RCCEx_CRS_IRQHandler():
     - HAL_RCCEx_CRS_SyncOkCallback()
     - HAL_RCCEx_CRS_SyncWarnCallback()
     - HAL_RCCEx_CRS_ExpectedSyncCallback()
     - HAL_RCCEx_CRS_ErrorCallback()
7. To force a SYNC EVENT, user can use the function HAL_RCCEx_CRSSoftwareSynchronizationGenerate(). This function can be called before calling HAL_RCCEx_CRSConfig (for instance in Systick handler)

This section contains the following APIs:

- ***HAL_RCCEx_CRSConfig()***
- ***HAL_RCCEx_CRSSoftwareSynchronizationGenerate()***
- ***HAL_RCCEx_CRSGetSynchronizationInfo()***
- ***HAL_RCCEx_CRSWaitSynchronization()***
- ***HAL_RCCEx_CRS_IRQHandler()***
- ***HAL_RCCEx_CRS_SyncOkCallback()***
- ***HAL_RCCEx_CRS_SyncWarnCallback()***
- ***HAL_RCCEx_CRS_ExpectedSyncCallback()***
- ***HAL_RCCEx_CRS_ErrorCallback()***

52.2.4 Detailed description of functions

HAL_RCCEx_PeriphCLKConfig

Function name	**HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)**
Function description	Initialize the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.
Parameters	• **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that contains a field PeriphClockSelection which can be a combination of the following values: – RCC_PERIPHCLK_RTC RTC peripheral clock – RCC_PERIPHCLK_ADC ADC peripheral clock – RCC_PERIPHCLK_I2C1 I2C1 peripheral clock – RCC_PERIPHCLK_I2C2 I2C2 peripheral clock – RCC_PERIPHCLK_I2C3 I2C3 peripheral clock – RCC_PERIPHCLK_I2C4 I2C4 peripheral clock (only for devices with I2C4) – RCC_PERIPHCLK_LPTIM1 LPTIM1 peripheral clock – RCC_PERIPHCLK_LPTIM2 LPTIM2 peripheral clock – RCC_PERIPHCLK_LPUART1 LPUART1 peripheral

clock
- RCC_PERIPHCLK_RNG RNG peripheral clock
- RCC_PERIPHCLK_SAI1 SAI1 peripheral clock
- RCC_PERIPHCLK_SAI2 SAI2 peripheral clock (only for devices with SAI2)
- RCC_PERIPHCLK_SDMMC1 SDMMC1 peripheral clock
- RCC_PERIPHCLK_USART1 USART1 peripheral clock
- RCC_PERIPHCLK_USART2 USART1 peripheral clock
- RCC_PERIPHCLK_USART3 USART1 peripheral clock
- RCC_PERIPHCLK_UART4 USART1 peripheral clock (only for devices with UART4)
- RCC_PERIPHCLK_UART5 USART1 peripheral clock (only for devices with UART5)
- RCC_PERIPHCLK_USB USB peripheral clock (only for devices with USB)
- RCC_PERIPHCLK_DFSDM1 DFSDM1 peripheral kernel clock (only for devices with DFSDM1)
- RCC_PERIPHCLK_DFSDM1AUDIO DFSDM1 peripheral audio clock (only for devices with DFSDM1)
- RCC_PERIPHCLK_LTDC LTDC peripheral clock (only for devices with LTDC)
- RCC_PERIPHCLK_DSI DSI peripheral clock (only for devices with DSI)
- RCC_PERIPHCLK_OSPI OctoSPI peripheral clock (only for devices with OctoSPI)

Return values	• **HAL:** status
Notes	• Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source: in this case the access to Backup domain is enabled.

HAL_RCCEx_GetPeriphCLKConfig

Function name	**void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)**
Function description	Get the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	• **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks(SAI1, SAI2, LPTIM1, LPTIM2, I2C1, I2C2, I2C3, I2C4, LPUART, USART1, USART2, USART3, UART4, UART5, RTC, ADCx, DFSDMx, SWPMI1, USB, SDMMC1 and RNG).
Return values	• **None:**

HAL_RCCEx_GetPeriphCLKFreq

Function name	**uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint32_t PeriphClk)**
Function description	Return the peripheral clock frequency for peripherals with clock source from PLLSAIs.
Parameters	• **PeriphClk:** Peripheral clock identifier This parameter can be

	one of the following values: – RCC_PERIPHCLK_RTC RTC peripheral clock – RCC_PERIPHCLK_ADC ADC peripheral clock – RCC_PERIPHCLK_I2C1 I2C1 peripheral clock – RCC_PERIPHCLK_I2C2 I2C2 peripheral clock – RCC_PERIPHCLK_I2C3 I2C3 peripheral clock – RCC_PERIPHCLK_I2C4 I2C4 peripheral clock (only for devices with I2C4) – RCC_PERIPHCLK_LPTIM1 LPTIM1 peripheral clock – RCC_PERIPHCLK_LPTIM2 LPTIM2 peripheral clock – RCC_PERIPHCLK_LPUART1 LPUART1 peripheral clock – RCC_PERIPHCLK_RNG RNG peripheral clock – RCC_PERIPHCLK_SAI1 SAI1 peripheral clock – RCC_PERIPHCLK_SAI2 SAI2 peripheral clock (only for devices with SAI2) – RCC_PERIPHCLK_SDMMC1 SDMMC1 peripheral clock – RCC_PERIPHCLK_USART1 USART1 peripheral clock – RCC_PERIPHCLK_USART2 USART1 peripheral clock – RCC_PERIPHCLK_USART3 USART1 peripheral clock – RCC_PERIPHCLK_UART4 USART1 peripheral clock (only for devices with UART4) – RCC_PERIPHCLK_UART5 USART1 peripheral clock (only for devices with UART5) – RCC_PERIPHCLK_USB USB peripheral clock (only for devices with USB) – RCC_PERIPHCLK_DFSDM1 DFSDM1 peripheral kernel clock (only for devices with DFSDM1) – RCC_PERIPHCLK_DFSDM1AUDIO DFSDM1 peripheral audio clock (only for devices with DFSDM1) – RCC_PERIPHCLK_LTDC LTDC peripheral clock (only for devices with LTDC) – RCC_PERIPHCLK_DSI DSI peripheral clock (only for devices with DSI) – RCC_PERIPHCLK_OSPI OctoSPI peripheral clock (only for devices with OctoSPI)
Return values	• **Frequency:** in Hz
Notes	• Return 0 if peripheral clock identifier not managed by this API

HAL_RCCEx_EnablePLLSAI1

Function name	**HAL_StatusTypeDef HAL_RCCEx_EnablePLLSAI1 (RCC_PLLSAI1InitTypeDef * PLLSAI1Init)**
Function description	Enable PLLSAI1.
Parameters	• **PLLSAI1Init:** pointer to an RCC_PLLSAI1InitTypeDef structure that contains the configuration information for the PLLSAI1
Return values	• **HAL:** status

HAL_RCCEx_DisablePLLSAI1

Function name	**HAL_StatusTypeDef HAL_RCCEx_DisablePLLSAI1 (void)**
Function description	Disable PLLSAI1.
Return values	• **HAL:** status

HAL_RCCEx_EnablePLLSAI2

Function name	**HAL_StatusTypeDef HAL_RCCEx_EnablePLLSAI2 (RCC_PLLSAI2InitTypeDef * PLLSAI2Init)**
Function description	Enable PLLSAI2.
Parameters	• **PLLSAI2Init:** pointer to an RCC_PLLSAI2InitTypeDef structure that contains the configuration information for the PLLSAI2
Return values	• **HAL:** status

HAL_RCCEx_DisablePLLSAI2

Function name	**HAL_StatusTypeDef HAL_RCCEx_DisablePLLSAI2 (void)**
Function description	Disable PLLISAI2.
Return values	• **HAL:** status

HAL_RCCEx_WakeUpStopCLKConfig

Function name	**void HAL_RCCEx_WakeUpStopCLKConfig (uint32_t WakeUpClk)**
Function description	Configure the oscillator clock source for wakeup from Stop and CSS backup clock.
Parameters	• **WakeUpClk:** Wakeup clock This parameter can be one of the following values: – RCC_STOP_WAKEUPCLOCK_MSI MSI oscillator selection – RCC_STOP_WAKEUPCLOCK_HSI HSI oscillator selection
Return values	• **None:**
Notes	• This function shall not be called after the Clock Security System on HSE has been enabled.

HAL_RCCEx_StandbyMSIRangeConfig

Function name	**void HAL_RCCEx_StandbyMSIRangeConfig (uint32_t MSIRange)**
Function description	Configure the MSI range after standby mode.
Parameters	• **MSIRange:** MSI range This parameter can be one of the following values: – RCC_MSIRANGE_4 Range 4 around 1 MHz – RCC_MSIRANGE_5 Range 5 around 2 MHz – RCC_MSIRANGE_6 Range 6 around 4 MHz (reset

value)
– RCC_MSIRANGE_7 Range 7 around 8 MHz

Return values	• **None:**
Notes	• After Standby its frequency can be selected between 4 possible values (1, 2, 4 or 8 MHz).

HAL_RCCEx_EnableLSECSS

Function name	**void HAL_RCCEx_EnableLSECSS (void)**
Function description	Enable the LSE Clock Security System.
Return values	• **None:**
Notes	• Prior to enable the LSE Clock Security System, LSE oscillator is to be enabled with HAL_RCC_OscConfig() and the LSE oscillator clock is to be selected as RTC clock with HAL_RCCEx_PeriphCLKConfig().

HAL_RCCEx_DisableLSECSS

Function name	**void HAL_RCCEx_DisableLSECSS (void)**
Function description	Disable the LSE Clock Security System.
Return values	• **None:**
Notes	• LSE Clock Security System can only be disabled after a LSE failure detection.

HAL_RCCEx_EnableLSECSS_IT

Function name	**void HAL_RCCEx_EnableLSECSS_IT (void)**
Function description	Enable the LSE Clock Security System Interrupt & corresponding EXTI line.
Return values	• **None:**
Notes	• LSE Clock Security System Interrupt is mapped on RTC EXTI line 19

HAL_RCCEx_LSECSS_IRQHandler

Function name	**void HAL_RCCEx_LSECSS_IRQHandler (void)**
Function description	Handle the RCC LSE Clock Security System interrupt request.
Return values	• **None:**

HAL_RCCEx_LSECSS_Callback

Function name	**void HAL_RCCEx_LSECSS_Callback (void)**
Function description	RCCEx LSE Clock Security System interrupt callback.
Return values	• **none:**

HAL_RCCEx_EnableLSCO

Function name	**void HAL_RCCEx_EnableLSCO (uint32_t LSCOSource)**
Function description	Select the Low Speed clock source to output on LSCO pin (PA2).
Parameters	• **LSCOSource:** specifies the Low Speed clock source to output. This parameter can be one of the following values: – RCC_LSCOSOURCE_LSI LSI clock selected as LSCO source – RCC_LSCOSOURCE_LSE LSE clock selected as LSCO source
Return values	• **None:**

HAL_RCCEx_DisableLSCO

Function name	**void HAL_RCCEx_DisableLSCO (void)**
Function description	Disable the Low Speed clock output.
Return values	• **None:**

HAL_RCCEx_EnableMSIPLLMode

Function name	**void HAL_RCCEx_EnableMSIPLLMode (void)**
Function description	Enable the PLL-mode of the MSI.
Return values	• **None:**
Notes	• Prior to enable the PLL-mode of the MSI for automatic hardware calibration LSE oscillator is to be enabled with HAL_RCC_OscConfig().

HAL_RCCEx_DisableMSIPLLMode

Function name	**void HAL_RCCEx_DisableMSIPLLMode (void)**
Function description	Disable the PLL-mode of the MSI.
Return values	• **None:**
Notes	• PLL-mode of the MSI is automatically reset when LSE oscillator is disabled.

HAL_RCCEx_CRSConfig

Function name	**void HAL_RCCEx_CRSConfig (RCC_CRSInitTypeDef * pInit)**
Function description	Start automatic synchronization for polling mode.
Parameters	• **pInit:** Pointer on RCC_CRSInitTypeDef structure
Return values	• **None:**

HAL_RCCEx_CRSSoftwareSynchronizationGenerate

Function name	**void HAL_RCCEx_CRSSoftwareSynchronizationGenerate (void)**

Function description	Generate the software synchronization event.
Return values	• **None:**

HAL_RCCEx_CRSGetSynchronizationInfo

Function name	**void HAL_RCCEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)**
Function description	Return synchronization info.
Parameters	• **pSynchroInfo:** Pointer on RCC_CRSSynchroInfoTypeDef structure
Return values	• **None:**

HAL_RCCEx_CRSWaitSynchronization

Function name	**uint32_t HAL_RCCEx_CRSWaitSynchronization (uint32_t Timeout)**
Function description	Wait for CRS Synchronization status.
Parameters	• **Timeout:** Duration of the timeout
Return values	• **Combination:** of Synchronization status This parameter can be a combination of the following values: – RCC_CRS_TIMEOUT – RCC_CRS_SYNCOK – RCC_CRS_SYNCWARN – RCC_CRS_SYNCERR – RCC_CRS_SYNCMISS – RCC_CRS_TRIMOVF
Notes	• Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency. • If Timeout set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.

HAL_RCCEx_CRS_IRQHandler

Function name	**void HAL_RCCEx_CRS_IRQHandler (void)**
Function description	Handle the Clock Recovery System interrupt request.
Return values	• **None:**

HAL_RCCEx_CRS_SyncOkCallback

Function name	**void HAL_RCCEx_CRS_SyncOkCallback (void)**
Function description	RCCEx Clock Recovery System SYNCOK interrupt callback.
Return values	• **none:**

HAL_RCCEx_CRS_SyncWarnCallback

Function name	**void HAL_RCCEx_CRS_SyncWarnCallback (void)**

Function description	RCCEx Clock Recovery System SYNCWARN interrupt callback.
Return values	• **none:**

HAL_RCCEx_CRS_ExpectedSyncCallback

Function name	**void HAL_RCCEx_CRS_ExpectedSyncCallback (void)**
Function description	RCCEx Clock Recovery System Expected SYNC interrupt callback.
Return values	• **none:**

HAL_RCCEx_CRS_ErrorCallback

Function name	**void HAL_RCCEx_CRS_ErrorCallback (uint32_t Error)**
Function description	RCCEx Clock Recovery System Error interrupt callback.
Parameters	• **Error:** Combination of Error status. This parameter can be a combination of the following values: – RCC_CRS_SYNCERR – RCC_CRS_SYNCMISS – RCC_CRS_TRIMOVF
Return values	• **none:**

52.3 RCCEx Firmware driver defines

52.3.1 RCCEx

ADC Clock Source

RCC_ADCCLKSOURCE_NONE

RCC_ADCCLKSOURCE_PLLSAI1

RCC_ADCCLKSOURCE_SYSCLK

RCCEx CRS ErrorLimitDefault

RCC_CRS_ERRORLIMIT_DEFAULT Default Frequency error limit

RCCEx CRS Extended Features

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_ENABLE	**Description:** • Enable the oscillator clock for frequency error counter. **Return value:** • None **Notes:** • when the CEN bit is set the CRS_CFGR register becomes write-protected.
__HAL_RCC_CRS_FREQ_ERROR_COUNTER_DISABLE	**Description:** • Disable the oscillator clock

for frequency error counter.

Return value:

- None

__HAL_RCC_CRS_AUTOMATIC_CALIB_ENABLE

Description:

- Enable the automatic hardware adjustement of TRIM bits.

Return value:

- None

Notes:

- When the AUTOTRIMEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_AUTOMATIC_CALIB_DISABLE

Description:

- Enable or disable the automatic hardware adjustement of TRIM bits.

Return value:

- None

__HAL_RCC_CRS_RELOADVALUE_CALCULATE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- None

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

RCCEx CRS Flags

RCC_CRS_FLAG_SYNCOK	SYNC event OK flag
RCC_CRS_FLAG_SYNCWARN	SYNC warning flag
RCC_CRS_FLAG_ERR	Error flag
RCC_CRS_FLAG_ESYNC	Expected SYNC flag
RCC_CRS_FLAG_SYNCERR	SYNC error
RCC_CRS_FLAG_SYNCMISS	SYNC missed
RCC_CRS_FLAG_TRIMOVF	Trimming overflow or underflow

RCCEx CRS FreqErrorDirection

RCC_CRS_FREQERRORDIR_UP	Upcounting direction, the actual frequency is above the target
RCC_CRS_FREQERRORDIR_DOWN	Downcounting direction, the actual frequency is below the target

RCCEx CRS HSI48CalibrationDefault

RCC_CRS_HSI48CALIBRATION_DEFAULT	The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency

RCCEx CRS Interrupt Sources

RCC_CRS_IT_SYNCOK	SYNC event OK
RCC_CRS_IT_SYNCWARN	SYNC warning
RCC_CRS_IT_ERR	Error
RCC_CRS_IT_ESYNC	Expected SYNC
RCC_CRS_IT_SYNCERR	SYNC error
RCC_CRS_IT_SYNCMISS	SYNC missed
RCC_CRS_IT_TRIMOVF	Trimming overflow or underflow

RCCEx CRS ReloadValueDefault

RCC_CRS_RELOADVALUE_DEFAULT	The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).

RCCEx CRS Status

RCC_CRS_NONE

RCC_CRS_TIMEOUT

RCC_CRS_SYNCOK

RCC_CRS_SYNCWARN

RCC_CRS_SYNCERR

RCC_CRS_SYNCMISS

RCC_CRS_TRIMOVF

RCCEx CRS SynchroDivider

RCC_CRS_SYNC_DIV1	Synchro Signal not divided (default)
RCC_CRS_SYNC_DIV2	Synchro Signal divided by 2
RCC_CRS_SYNC_DIV4	Synchro Signal divided by 4
RCC_CRS_SYNC_DIV8	Synchro Signal divided by 8
RCC_CRS_SYNC_DIV16	Synchro Signal divided by 16
RCC_CRS_SYNC_DIV32	Synchro Signal divided by 32
RCC_CRS_SYNC_DIV64	Synchro Signal divided by 64
RCC_CRS_SYNC_DIV128	Synchro Signal divided by 128

RCCEx CRS SynchroPolarity

RCC_CRS_SYNC_POLARITY_RISING	Synchro Active on rising edge (default)
RCC_CRS_SYNC_POLARITY_FALLING	Synchro Active on falling edge

RCCEx CRS SynchroSource

RCC_CRS_SYNC_SOURCE_GPIO	Synchro Signal source GPIO
RCC_CRS_SYNC_SOURCE_LSE	Synchro Signal source LSE
RCC_CRS_SYNC_SOURCE_USB	Synchro Signal source USB SOF (default)

DFSDM1 Audio Clock Source

RCC_DFSDM1AUDIOCLKSOURCE_SAI1

RCC_DFSDM1AUDIOCLKSOURCE_HSI

RCC_DFSDM1AUDIOCLKSOURCE_MSI

DFSDM1 Clock Source

RCC_DFSDM1CLKSOURCE_PCLK2

RCC_DFSDM1CLKSOURCE_SYSCLK

DSI Clock Source

RCC_DSICLKSOURCE_DSIPHY

RCC_DSICLKSOURCE_PLLSAI2

RCCEx Exported Macros

__HAL_RCC_PLLSAI1_CONFIG

Description:

- Macro to configure the PLLSAI1 clock multiplication and division factors.

Parameters:

- __PLLSAI1M__: specifies the division factor of PLLSAI1 input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16.

- __PLLSAI1N__: specifies the multiplication factor for PLLSAI1 VCO output clock. This parameter must be a number between 8 and 86.
- __PLLSAI1P__: specifies the division factor for SAI clock. This parameter must be a number in the range (7 or 17) for STM32L47xxx/L48xxx else (2 to 31). SAI1 clock frequency = f(PLLSAI1) / PLLSAI1P
- __PLLSAI1Q__: specifies the division factor for USB/RNG/SDMMC1 clock. This parameter must be in the range (2, 4, 6 or 8). USB/RNG/SDMMC1 clock frequency = f(PLLSAI1) / PLLSAI1Q
- __PLLSAI1R__: specifies the division factor for SAR ADC clock. This parameter must be in the range (2, 4, 6 or 8). ADC clock frequency = f(PLLSAI1) / PLLSAI1R

Return value:

- None

Notes:

- This function must be used only when the PLLSAI1 is disabled. PLLSAI1 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)
- You have to set the PLLSAI1N parameter correctly to ensure that the VCO output frequency is between 64 and 344 MHz. PLLSAI1 clock frequency = f(PLLSAI1) multiplied by PLLSAI1N

__HAL_RCC_PLLSAI1_MULN_CONFIG

Description:

- Macro to configure the PLLSAI1 clock multiplication factor N.

Parameters:

- __PLLSAI1N__: specifies the multiplication factor for PLLSAI1 VCO output clock. This parameter must be a number between 8 and 86.

Return value:

- None

Notes:

- This function must be used only when the PLLSAI1 is disabled. PLLSAI1 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)
- You have to set the PLLSAI1N parameter correctly to ensure that the VCO output

frequency is between 64 and 344 MHz. Use to set PLLSAI1 clock frequency = f(PLLSAI1) multiplied by PLLSAI1N

__HAL_RCC_PLLSAI1_DIVM_CONFIG

Description:

- Macro to configure the PLLSAI1 input clock division factor M.

Parameters:

- __PLLSAI1M__: specifies the division factor for PLLSAI1 clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16.

Return value:

- None

Notes:

- This function must be used only when the PLLSAI1 is disabled. PLLSAI1 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)

__HAL_RCC_PLLSAI1_DIVP_CONFIG

Description:

- Macro to configure the PLLSAI1 clock division factor P.

Parameters:

- __PLLSAI1P__: specifies the division factor for SAI clock. This parameter must be a number in the range (7 or 17) for STM32L47xxx/L48xxx else (2 to 31). Use to set SAI1 clock frequency = f(PLLSAI1) / PLLSAI1P

Return value:

- None

Notes:

- This function must be used only when the PLLSAI1 is disabled. PLLSAI1 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)

__HAL_RCC_PLLSAI1_DIVQ_CONFIG

Description:

- Macro to configure the PLLSAI1 clock division factor Q.

Parameters:

- __PLLSAI1Q__: specifies the division factor for USB/RNG/SDMMC1 clock. This parameter must be in the range (2, 4, 6 or

8). Use to set USB/RNG/SDMMC1 clock frequency = f(PLLSAI1) / PLLSAI1Q

Return value:

- None

Notes:

- This function must be used only when the PLLSAI1 is disabled. PLLSAI1 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)

__HAL_RCC_PLLSAI1_DIVR_CONFIG

Description:

- Macro to configure the PLLSAI1 clock division factor R.

Parameters:

- __PLLSAI1R__: specifies the division factor for ADC clock. This parameter must be in the range (2, 4, 6 or 8) Use to set ADC clock frequency = f(PLLSAI1) / PLLSAI1R

Return value:

- None

Notes:

- This function must be used only when the PLLSAI1 is disabled. PLLSAI1 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)

__HAL_RCC_PLLSAI1_ENABLE

Description:

- Macros to enable or disable the PLLSAI1.

Return value:

- None

Notes:

- The PLLSAI1 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLLSAI1_DISABLE

__HAL_RCC_PLLSAI1CLKOUT_ENABLE

Description:

- Macros to enable or disable each clock output (PLLSAI1_SAI1, PLLSAI1_USB2 and PLLSAI1_ADC1).

Parameters:

- __PLLSAI1_CLOCKOUT__: specifies the PLLSAI1 clock to be output. This parameter

can be one or a combination of the following values:
  - RCC_PLLSAI1_SAI1CLK This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI interface in case.
  - RCC_PLLSAI1_48M2CLK This clock is used to generate the clock for the USB OTG FS (48 MHz), the random number generator (<=48 MHz) and the SDIO (<= 48 MHz).
  - RCC_PLLSAI1_ADC1CLK Clock used to clock ADC peripheral.

Return value:

- None

Notes:

- Enabling and disabling those clocks can be done without the need to stop the PLL. This is mainly used to save Power.

__HAL_RCC_PLLSAI1CLKOUT_DISABLE

__HAL_RCC_GET_PLLSAI1CLKOUT_CONFIG

Description:

- Macro to get clock output enable status (PLLSAI1_SAI1, PLLSAI1_USB2 and PLLSAI1_ADC1).

Parameters:

- __PLLSAI1_CLOCKOUT__: specifies the PLLSAI1 clock to be output. This parameter can be one of the following values:
  - RCC_PLLSAI1_SAI1CLK This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI interface in case.
  - RCC_PLLSAI1_48M2CLK This clock is used to generate the clock for the USB OTG FS (48 MHz), the random number generator (<=48 MHz) and the SDIO (<= 48 MHz).
  - RCC_PLLSAI1_ADC1CLK Clock used to clock ADC peripheral.

Return value:

- SET: / RESET

__HAL_RCC_PLLSAI2_CONFIG

Description:

- Macro to configure the PLLSAI2 clock multiplication and division factors.

Parameters:

- __PLLSAI2M__: specifies the division

factor of PLLSAI2 input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16.

- __PLLSAI2N__: specifies the multiplication factor for PLLSAI2 VCO output clock. This parameter must be a number between 8 and 86.
- __PLLSAI2P__: specifies the division factor for SAI clock. This parameter must be a number in the range (7 or 17) for STM32L47xxx/L48xxx else (2 to 31). SAI2 clock frequency = f(PLLSAI2) / PLLSAI2P
- __PLLSAI2Q__: specifies the division factor for DSI clock. This parameter must be in the range (2, 4, 6 or 8). DSI clock frequency = f(PLLSAI2) / PLLSAI2Q
- __PLLSAI2R__: specifies the division factor for SAR ADC clock. This parameter must be in the range (2, 4, 6 or 8).

Return value:

- None

Notes:

- This function must be used only when the PLLSAI2 is disabled. PLLSAI2 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)
- You have to set the PLLSAI2N parameter correctly to ensure that the VCO output frequency is between 64 and 344 MHz.

__HAL_RCC_PLLSAI2_MULN_CONFIG

Description:

- Macro to configure the PLLSAI2 clock multiplication factor N.

Parameters:

- __PLLSAI2N__: specifies the multiplication factor for PLLSAI2 VCO output clock. This parameter must be a number between 8 and 86.

Return value:

- None

Notes:

- This function must be used only when the PLLSAI2 is disabled. PLLSAI2 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)
- You have to set the PLLSAI2N parameter correctly to ensure that the VCO output frequency is between 64 and 344 MHz.

PLLSAI1 clock frequency = f(PLLSAI1) multiplied by PLLSAI2N

__HAL_RCC_PLLSAI2_DIVM_CONFIG

Description:

- Macro to configure the PLLSAI2 input clock division factor M.

Parameters:

- __PLLSAI2M__: specifies the division factor for PLLSAI2 clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16.

Return value:

- None

Notes:

- This function must be used only when the PLLSAI2 is disabled. PLLSAI2 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)

__HAL_RCC_PLLSAI2_DIVP_CONFIG

Description:

- Macro to configure the PLLSAI2 clock division factor P.

Parameters:

- __PLLSAI2P__: specifies the division factor. This parameter must be a number in the range (7 or 17). Use to set SAI2 clock frequency = f(PLLSAI2) / __PLLSAI2P__

Return value:

- None

Notes:

- This function must be used only when the PLLSAI2 is disabled. PLLSAI2 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)

__HAL_RCC_PLLSAI2_DIVQ_CONFIG

Description:

- Macro to configure the PLLSAI2 clock division factor Q.

Parameters:

- __PLLSAI2Q__: specifies the division factor for USB/RNG/SDMMC1 clock. This parameter must be in the range (2, 4, 6 or 8). Use to set USB/RNG/SDMMC1 clock frequency = f(PLLSAI2) / PLLSAI2Q

Return value:

- None

Notes:

- This function must be used only when the PLLSAI2 is disabled. PLLSAI2 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)

__HAL_RCC_PLLSAI2_DIVR_CONFIG

Description:

- Macro to configure the PLLSAI2 clock division factor R.

Parameters:

- __PLLSAI2R__: specifies the division factor. This parameter must be in the range (2, 4, 6 or 8). Use to set ADC clock frequency = f(PLLSAI2) / __PLLSAI2R__

Return value:

- None

Notes:

- This function must be used only when the PLLSAI2 is disabled. PLLSAI2 clock source is common with the main PLL (configured through __HAL_RCC_PLL_CONFIG() macro)

__HAL_RCC_PLLSAI2_ENABLE

Description:

- Macros to enable or disable the PLLSAI2.

Return value:

- None

Notes:

- The PLLSAI2 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLLSAI2_DISABLE

__HAL_RCC_PLLSAI2CLKOUT_ENABLE

Description:

- Macros to enable or disable each clock output (PLLSAI2_SAI2, PLLSAI2_ADC2 and RCC_PLLSAI2_DSICLK).

Parameters:

- __PLLSAI2_CLOCKOUT__: specifies the PLLSAI2 clock to be output. This parameter can be one or a combination of the following values:
  - RCC_PLLSAI2_SAI2CLK This clock is

used to generate an accurate clock to achieve high-quality audio performance on SAI interface in case.
    - RCC_PLLSAI2_DSICLK Clock used to clock DSI peripheral.

Return value:

- None

Notes:

- Enabling and disabling those clocks can be done without the need to stop the PLL. This is mainly used to save Power.

__HAL_RCC_PLLSAI2CLKOUT_DISABLE

__HAL_RCC_GET_PLLSAI2CLKOUT_CONFIG

Description:

- Macro to get clock output enable status (PLLSAI2_SAI2, PLLSAI2_ADC2 and RCC_PLLSAI2_DSICLK).

Parameters:

- __PLLSAI2_CLOCKOUT__: specifies the PLLSAI2 clock to be output. This parameter can be one of the following values:
    - RCC_PLLSAI2_SAI2CLK This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI interface in case.
    - RCC_PLLSAI2_DSICLK Clock used to clock DSI peripheral.

Return value:

- SET: / RESET

__HAL_RCC_SAI1_CONFIG

Description:

- Macro to configure the SAI1 clock source.

Parameters:

- __SAI1_CLKSOURCE__: defines the SAI1 clock source. This clock is derived from the PLLSAI1, system PLL or external clock (through a dedicated pin). This parameter can be one of the following values:
    - RCC_SAI1CLKSOURCE_PLLSAI1 SAI1 clock = PLLSAI1 "P" clock (PLLSAI1CLK)
    - RCC_SAI1CLKSOURCE_PLL SAI1 clock = PLL "P" clock (PLLSAI3CLK if PLLSAI2 exists, else PLLSAI2CLK)
    - RCC_SAI1CLKSOURCE_PIN SAI1 clock = External Clock (SAI1_EXTCLK)
    - RCC_SAI1CLKSOURCE_HSI SAI1

clock = HSI16

Return value:

- None

__HAL_RCC_GET_SAI1_SOURCE

Description:

- Macro to get the SAI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI1CLKSOURCE_PLLSAI1 SAI1 clock = PLLSAI1 "P" clock (PLLSAI1CLK)
  - RCC_SAI1CLKSOURCE_PLL SAI1 clock = PLL "P" clock (PLLSAI3CLK if PLLSAI2 exists, else PLLSAI2CLK)
  - RCC_SAI1CLKSOURCE_PIN SAI1 clock = External Clock (SAI1_EXTCLK)

Notes:

- Despite returned values RCC_SAI1CLKSOURCE_PLLSAI1 or RCC_SAI1CLKSOURCE_PLL, HSI16 is automatically set as SAI1 clock source when PLLs are disabled for devices without PLLSAI2.

__HAL_RCC_SAI2_CONFIG

Description:

- Macro to configure the SAI2 clock source.

Parameters:

- __SAI2_CLKSOURCE__: defines the SAI2 clock source. This clock is derived from the PLLSAI2, system PLL or external clock (through a dedicated pin). This parameter can be one of the following values:
  - RCC_SAI2CLKSOURCE_PLLSAI1 SAI2 clock = PLLSAI1 "P" clock (PLLSAI1CLK)
  - RCC_SAI2CLKSOURCE_PLLSAI2 SAI2 clock = PLLSAI2 "P" clock (PLLSAI2CLK)
  - RCC_SAI2CLKSOURCE_PLL SAI2 clock = PLL "P" clock (PLLSAI3CLK)
  - RCC_SAI2CLKSOURCE_PIN SAI2 clock = External Clock (SAI2_EXTCLK)
  - RCC_SAI2CLKSOURCE_HSI SAI2 clock = HSI16

Return value:

- None

__HAL_RCC_GET_SAI2_SOURCE

Description:

- Macro to get the SAI2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI2CLKSOURCE_PLLSAI1 SAI2 clock = PLLSAI1 "P" clock (PLLSAI1CLK)
  - RCC_SAI2CLKSOURCE_PLLSAI2 SAI2 clock = PLLSAI2 "P" clock (PLLSAI2CLK)
  - RCC_SAI2CLKSOURCE_PLL SAI2 clock = PLL "P" clock (PLLSAI3CLK)
  - RCC_SAI2CLKSOURCE_PIN SAI2 clock = External Clock (SAI2_EXTCLK)

__HAL_RCC_I2C1_CONFIG

Description:

- Macro to configure the I2C1 clock (I2C1CLK).

Parameters:

- __I2C1_CLKSOURCE__: specifies the I2C1 clock source. This parameter can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK System Clock selected as I2C1 clock

Return value:

- None

__HAL_RCC_GET_I2C1_SOURCE

Description:

- Macro to get the I2C1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK System Clock selected as I2C1 clock

__HAL_RCC_I2C2_CONFIG

Description:

- Macro to configure the I2C2 clock (I2C2CLK).

Parameters:

- __I2C2_CLKSOURCE__: specifies the I2C2 clock source. This parameter can be one of the following values:
  - RCC_I2C2CLKSOURCE_PCLK1 PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_HSI HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_SYSCLK System Clock selected as I2C2 clock

Return value:

- None

__HAL_RCC_GET_I2C2_SOURCE

Description:

- Macro to get the I2C2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C2CLKSOURCE_PCLK1 PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_HSI HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_SYSCLK System Clock selected as I2C2 clock

__HAL_RCC_I2C3_CONFIG

Description:

- Macro to configure the I2C3 clock (I2C3CLK).

Parameters:

- __I2C3_CLKSOURCE__: specifies the I2C3 clock source. This parameter can be one of the following values:
  - RCC_I2C3CLKSOURCE_PCLK1 PCLK1 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI HSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_SYSCLK System Clock selected as I2C3 clock

Return value:

- None

__HAL_RCC_GET_I2C3_SOURCE

Description:

- Macro to get the I2C3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C3CLKSOURCE_PCLK1 PCLK1 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI HSI selected as I2C3 clock

- RCC_I2C3CLKSOURCE_SYSCLK System Clock selected as I2C3 clock

__HAL_RCC_I2C4_CONFIG

Description:

- Macro to configure the I2C4 clock (I2C4CLK).

Parameters:

- __I2C4_CLKSOURCE__: specifies the I2C4 clock source. This parameter can be one of the following values:
  - RCC_I2C4CLKSOURCE_PCLK1 PCLK1 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_HSI HSI selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_SYSCLK System Clock selected as I2C4 clock

Return value:

- None

__HAL_RCC_GET_I2C4_SOURCE

Description:

- Macro to get the I2C4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C4CLKSOURCE_PCLK1 PCLK1 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_HSI HSI selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_SYSCLK System Clock selected as I2C4 clock

__HAL_RCC_USART1_CONFIG

Description:

- Macro to configure the USART1 clock (USART1CLK).

Parameters:

- __USART1_CLKSOURCE__: specifies the USART1 clock source. This parameter can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK2 selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE SE selected as USART1 clock

Return value:

- None

__HAL_RCC_GET_USART1_SOURCE

Description:

- Macro to get the USART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK2 selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

__HAL_RCC_USART2_CONFIG

Description:

- Macro to configure the USART2 clock (USART2CLK).

Parameters:

- __USART2_CLKSOURCE__: specifies the USART2 clock source. This parameter can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_SYSCLK System Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

Return value:

- None

__HAL_RCC_GET_USART2_SOURCE

Description:

- Macro to get the USART2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_SYSCLK System Clock selected as USART2 clock

- RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

__HAL_RCC_USART3_CONFIG

Description:

- Macro to configure the USART3 clock (USART3CLK).

Parameters:

- __USART3_CLKSOURCE__: specifies the USART3 clock source. This parameter can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1 PCLK1 selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_SYSCLK System Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock

Return value:

- None

__HAL_RCC_GET_USART3_SOURCE

Description:

- Macro to get the USART3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1 PCLK1 selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_SYSCLK System Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock

__HAL_RCC_UART4_CONFIG

Description:

- Macro to configure the UART4 clock (UART4CLK).

Parameters:

- __UART4_CLKSOURCE__: specifies the UART4 clock source. This parameter can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1 PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_SYSCLK System Clock selected as UART4

clock
- RCC_UART4CLKSOURCE_LSE LSE selected as UART4 clock

Return value:

- None

__HAL_RCC_GET_UART4_SOURCE

Description:

- Macro to get the UART4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1 PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_SYSCLK System Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE LSE selected as UART4 clock

__HAL_RCC_UART5_CONFIG

Description:

- Macro to configure the UART5 clock (UART5CLK).

Parameters:

- __UART5_CLKSOURCE__: specifies the UART5 clock source. This parameter can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1 PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_SYSCLK System Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE LSE selected as UART5 clock

Return value:

- None

__HAL_RCC_GET_UART5_SOURCE

Description:

- Macro to get the UART5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1 PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI HSI selected as UART5 clock

- RCC_UART5CLKSOURCE_SYSCLK System Clock selected as UART5 clock
- RCC_UART5CLKSOURCE_LSE LSE selected as UART5 clock

__HAL_RCC_LPUART1_CONFIG

Description:

- Macro to configure the LPUART1 clock (LPUART1CLK).

Parameters:

- __LPUART1_CLKSOURCE__: specifies the LPUART1 clock source. This parameter can be one of the following values:
  - RCC_LPUART1CLKSOURCE_PCLK1 PCLK1 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_SYSCLK System Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock

Return value:

- None

__HAL_RCC_GET_LPUART1_SOURCE

Description:

- Macro to get the LPUART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPUART1CLKSOURCE_PCLK1 PCLK1 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_SYSCLK System Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock

__HAL_RCC_LPTIM1_CONFIG

Description:

- Macro to configure the LPTIM1 clock (LPTIM1CLK).

Parameters:

- __LPTIM1_CLKSOURCE__: specifies the LPTIM1 clock source. This parameter can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_PCLK1 PCLK1 selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI HSI

selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_HSI LSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSE LSE selected as LPTIM1 clock

Return value:

- None

__HAL_RCC_GET_LPTIM1_SOURCE

Description:

- Macro to get the LPTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_PCLK1 PCLK1 selected as LPUART1 clock
  - RCC_LPTIM1CLKSOURCE_LSI HSI selected as LPUART1 clock
  - RCC_LPTIM1CLKSOURCE_HSI System Clock selected as LPUART1 clock
  - RCC_LPTIM1CLKSOURCE_LSE LSE selected as LPUART1 clock

__HAL_RCC_LPTIM2_CONFIG

Description:

- Macro to configure the LPTIM2 clock (LPTIM2CLK).

Parameters:

- __LPTIM2_CLKSOURCE__: specifies the LPTIM2 clock source. This parameter can be one of the following values:
  - RCC_LPTIM2CLKSOURCE_PCLK1 PCLK1 selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSI HSI selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_HSI LSI selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSE LSE selected as LPTIM2 clock

Return value:

- None

__HAL_RCC_GET_LPTIM2_SOURCE

Description:

- Macro to get the LPTIM2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM2CLKSOURCE_PCLK1 PCLK1 selected as LPUART1 clock
  - RCC_LPTIM2CLKSOURCE_LSI HSI

selected as LPUART1 clock
- RCC_LPTIM2CLKSOURCE_HSI System Clock selected as LPUART1 clock
- RCC_LPTIM2CLKSOURCE_LSE LSE selected as LPUART1 clock

__HAL_RCC_SDMMC1_CONFIG

Description:

- Macro to configure the SDMMC1 clock.

Parameters:

- __SDMMC1_CLKSOURCE__: specifies the SDMMC1 clock source. This parameter can be one of the following values:
  - RCC_SDMMC1CLKSOURCE_HSI48 HSI48 selected as SDMMC1 clock for devices with HSI48
  - RCC_SDMMC1CLKSOURCE_MSI MSI selected as SDMMC1 clock
  - RCC_SDMMC1CLKSOURCE_PLLSAI1 PLLSAI1 Clock selected as SDMMC1 clock
  - RCC_SDMMC1CLKSOURCE_PLL PLL Clock selected as SDMMC1 clock

Return value:

- None

__HAL_RCC_GET_SDMMC1_SOURCE

Description:

- Macro to get the SDMMC1 clock.

Return value:

- The: clock source can be one of the following values:
  - RCC_SDMMC1CLKSOURCE_HSI48 HSI48 selected as SDMMC1 clock for devices with HSI48
  - RCC_SDMMC1CLKSOURCE_MSI MSI selected as SDMMC1 clock
  - RCC_SDMMC1CLKSOURCE_PLLSAI1 PLLSAI1 "Q" clock (PLL48M2CLK) selected as SDMMC1 clock
  - RCC_SDMMC1CLKSOURCE_PLL PLL "Q" clock (PLL48M1CLK) selected as SDMMC1 clock

__HAL_RCC_RNG_CONFIG

Description:

- Macro to configure the RNG clock.

Parameters:

- __RNG_CLKSOURCE__: specifies the RNG clock source. This parameter can be one of the following values:
  - RCC_RNGCLKSOURCE_MSI MSI

selected as RNG clock
  - RCC_RNGCLKSOURCE_PLLSAI1 PLLSAI1 Clock selected as RNG clock
  - RCC_RNGCLKSOURCE_PLL PLL Clock selected as RNG clock

Return value:

- None

Notes:

- USB, RNG and SDMMC1 peripherals share the same 48MHz clock source.

__HAL_RCC_GET_RNG_SOURCE

Description:

- Macro to get the RNG clock.

Return value:

- The: clock source can be one of the following values:
  - RCC_RNGCLKSOURCE_MSI MSI selected as RNG clock
  - RCC_RNGCLKSOURCE_PLLSAI1 PLLSAI1 "Q" clock (PLL48M2CLK) selected as RNG clock
  - RCC_RNGCLKSOURCE_PLL PLL "Q" clock (PLL48M1CLK) selected as RNG clock

__HAL_RCC_USB_CONFIG

Description:

- Macro to configure the USB clock (USBCLK).

Parameters:

- __USB_CLKSOURCE__: specifies the USB clock source. This parameter can be one of the following values:
  - RCC_USBCLKSOURCE_MSI MSI selected as USB clock
  - RCC_USBCLKSOURCE_PLLSAI1 PLLSAI1 "Q" clock (PLL48M2CLK) selected as USB clock
  - RCC_USBCLKSOURCE_PLL PLL "Q" clock (PLL48M1CLK) selected as USB clock

Return value:

- None

Notes:

- USB, RNG and SDMMC1 peripherals share the same 48MHz clock source.

__HAL_RCC_GET_USB_SOURCE

Description:

- Macro to get the USB clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_USBCLKSOURCE_MSI MSI selected as USB clock
  – RCC_USBCLKSOURCE_PLLSAI1 PLLSAI1 "Q" clock (PLL48M2CLK) selected as USB clock
  – RCC_USBCLKSOURCE_PLL PLL "Q" clock (PLL48M1CLK) selected as USB clock

__HAL_RCC_ADC_CONFIG

Description:

- Macro to configure the ADC interface clock.

Parameters:

- __ADC_CLKSOURCE__: specifies the ADC digital interface clock source. This parameter can be one of the following values:
  – RCC_ADCCLKSOURCE_NONE No clock selected as ADC clock
  – RCC_ADCCLKSOURCE_PLLSAI1 PLLSAI1 Clock selected as ADC clock
  – RCC_ADCCLKSOURCE_SYSCLK System Clock selected as ADC clock

Return value:

- None

__HAL_RCC_GET_ADC_SOURCE

Description:

- Macro to get the ADC clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_ADCCLKSOURCE_NONE No clock selected as ADC clock
  – RCC_ADCCLKSOURCE_PLLSAI1 PLLSAI1 Clock selected as ADC clock
  – RCC_ADCCLKSOURCE_SYSCLK System Clock selected as ADC clock

__HAL_RCC_DFSDM1_CONFIG

Description:

- Macro to configure the DFSDM1 clock.

Parameters:

- __DFSDM1_CLKSOURCE__: specifies the DFSDM1 clock source. This parameter can be one of the following values:
  – RCC_DFSDM1CLKSOURCE_PCLK2 PCLK2 Clock selected as DFSDM1 clock
  – RCC_DFSDM1CLKSOURCE_SYSCL

K System Clock selected as DFSDM1 clock

Return value:

- None

__HAL_RCC_GET_DFSDM1_SOURCE

Description:

- Macro to get the DFSDM1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_DFSDM1CLKSOURCE_PCLK2 PCLK2 Clock selected as DFSDM1 clock
  - RCC_DFSDM1CLKSOURCE_SYSCLK System Clock selected as DFSDM1 clock

__HAL_RCC_DFSDM1AUDIO_CONFIG

Description:

- Macro to configure the DFSDM1 audio clock.

Parameters:

- __DFSDM1AUDIO_CLKSOURCE__: specifies the DFSDM1 audio clock source. This parameter can be one of the following values:
  - RCC_DFSDM1AUDIOCLKSOURCE_SAI1 SAI1 clock selected as DFSDM1 audio clock
  - RCC_DFSDM1AUDIOCLKSOURCE_HSI HSI clock selected as DFSDM1 audio clock
  - RCC_DFSDM1AUDIOCLKSOURCE_MSI MSI clock selected as DFSDM1 audio clock

Return value:

- None

__HAL_RCC_GET_DFSDM1AUDIO_SOURCE

Description:

- Macro to get the DFSDM1 audio clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_DFSDM1AUDIOCLKSOURCE_SAI1 SAI1 clock selected as DFSDM1 audio clock
  - RCC_DFSDM1AUDIOCLKSOURCE_HSI HSI clock selected as DFSDM1 audio clock

  - RCC_DFSDM1AUDIOCLKSOURCE_MSI MSI clock selected as DFSDM1 audio clock

__HAL_RCC_LTDC_CONFIG

Description:

- Macro to configure the LTDC clock.

Parameters:

- __LTDC_CLKSOURCE__: specifies the DSI clock source. This parameter can be one of the following values:
  - RCC_LTDCCLKSOURCE_PLLSAI2_DIV2 PLLSAI2 divider R divided by 2 clock selected as LTDC clock
  - RCC_LTDCCLKSOURCE_PLLSAI2_DIV4 PLLSAI2 divider R divided by 4 clock selected as LTDC clock
  - RCC_LTDCCLKSOURCE_PLLSAI2_DIV8 PLLSAI2 divider R divided by 8 clock selected as LTDC clock
  - RCC_LTDCCLKSOURCE_PLLSAI2_DIV16 PLLSAI2 divider R divided by 16 clock selected as LTDC clock

Return value:

- None

__HAL_RCC_GET_LTDC_SOURCE

Description:

- Macro to get the LTDC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LTDCCLKSOURCE_PLLSAI2_DIV2 PLLSAI2 divider R divided by 2 clock selected as LTDC clock
  - RCC_LTDCCLKSOURCE_PLLSAI2_DIV4 PLLSAI2 divider R divided by 4 clock selected as LTDC clock
  - RCC_LTDCCLKSOURCE_PLLSAI2_DIV8 PLLSAI2 divider R divided by 8 clock selected as LTDC clock
  - RCC_LTDCCLKSOURCE_PLLSAI2_DIV16 PLLSAI2 divider R divided by 16 clock selected as LTDC clock

__HAL_RCC_DSI_CONFIG

Description:

- Macro to configure the DSI clock.

Parameters:

- __DSI_CLKSOURCE__: specifies the DSI clock source. This parameter can be one of the following values:
  - RCC_DSICLKSOURCE_DSIPHY DSI-

PHY clock selected as DSI clock
    - RCC_DSICLKSOURCE_PLLSAI2 PLLSAI2 R divider clock selected as DSI clock

Return value:

- None

__HAL_RCC_GET_DSI_SOURCE

Description:

- Macro to get the DSI clock source.

Return value:

- The: clock source can be one of the following values:
    - RCC_DSICLKSOURCE_DSIPHY DSI-PHY clock selected as DSI clock
    - RCC_DSICLKSOURCE_PLLSAI2 PLLSAI2 R divider clock selected as DSI clock

__HAL_RCC_OSPI_CONFIG

Description:

- Macro to configure the OctoSPI clock.

Parameters:

- __OSPI_CLKSOURCE__: specifies the OctoSPI clock source. This parameter can be one of the following values:
    - RCC_OSPICLKSOURCE_SYSCLK System Clock selected as OctoSPI clock
    - RCC_OSPICLKSOURCE_MSI MSI clock selected as OctoSPI clock
    - RCC_OSPICLKSOURCE_PLL PLL Q divider clock selected as OctoSPI clock

Return value:

- None

__HAL_RCC_GET_OSPI_SOURCE

Description:

- Macro to get the OctoSPI clock source.

Return value:

- The: clock source can be one of the following values:
    - RCC_OSPICLKSOURCE_SYSCLK System Clock selected as OctoSPI clock
    - RCC_OSPICLKSOURCE_MSI MSI clock selected as OctoSPI clock
    - RCC_OSPICLKSOURCE_PLL PLL Q divider clock selected as OctoSPI clock

RCC LSE CSS external interrupt line

RCC_EXTI_LINE_LSECSS	External interrupt line 19 connected to the LSE CSS EXTI Line

Flags Interrupts Management

__HAL_RCC_PLLSAI1_ENABLE_IT	**Description:** • Enable PLLSAI1RDY interrupt. **Return value:** • None
__HAL_RCC_PLLSAI1_DISABLE_IT	**Description:** • Disable PLLSAI1RDY interrupt. **Return value:** • None
__HAL_RCC_PLLSAI1_CLEAR_IT	**Description:** • Clear the PLLSAI1RDY interrupt pending bit. **Return value:** • None
__HAL_RCC_PLLSAI1_GET_IT_SOURCE	**Description:** • Check whether PLLSAI1RDY interrupt has occurred or not. **Return value:** • TRUE: or FALSE.
__HAL_RCC_PLLSAI1_GET_FLAG	**Description:** • Check whether the PLLSAI1RDY flag is set or not. **Return value:** • TRUE: or FALSE.
__HAL_RCC_PLLSAI2_ENABLE_IT	**Description:** • Enable PLLSAI2RDY interrupt. **Return value:** • None
__HAL_RCC_PLLSAI2_DISABLE_IT	**Description:** • Disable PLLSAI2RDY interrupt. **Return value:** • None
__HAL_RCC_PLLSAI2_CLEAR_IT	**Description:** • Clear the PLLSAI2RDY interrupt

pending bit.

Return value:

- None

__HAL_RCC_PLLSAI2_GET_IT_SOURCE

Description:

- Check whether the PLLSAI2RDY interrupt has occurred or not.

Return value:

- TRUE: or FALSE.

__HAL_RCC_PLLSAI2_GET_FLAG

Description:

- Check whether the PLLSAI2RDY flag is set or not.

Return value:

- TRUE: or FALSE.

__HAL_RCC_LSECSS_EXTI_ENABLE_IT

Description:

- Enable the RCC LSE CSS Extended Interrupt Line.

Return value:

- None

__HAL_RCC_LSECSS_EXTI_DISABLE_IT

Description:

- Disable the RCC LSE CSS Extended Interrupt Line.

Return value:

- None

__HAL_RCC_LSECSS_EXTI_ENABLE_EVENT

Description:

- Enable the RCC LSE CSS Event Line.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_EVENT

Description:

- Disable the RCC LSE CSS Event Line.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the RCC LSE CSS Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_ FALLING_EDGE	**Description:** • Disable the RCC LSE CSS Extended Interrupt Falling Trigger. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_ENABLE_ RISING_ EDGE	**Description:** • Enable the RCC LSE CSS Extended Interrupt Rising Trigger. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_DISABLE_ RISING_ EDGE	**Description:** • Disable the RCC LSE CSS Extended Interrupt Rising Trigger. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_ENABLE_ RISING_ FALLING_EDGE	**Description:** • Enable the RCC LSE CSS Extended Interrupt Rising & Falling Trigger. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_DISABLE_ RISING_ FALLING_EDGE	**Description:** • Disable the RCC LSE CSS Extended Interrupt Rising & Falling Trigger. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_GET_FLAG	**Description:** • Check whether the specified RCC LSE CSS EXTI interrupt flag is set or not. **Return value:** • EXTI: RCC LSE CSS Line Status.
__HAL_RCC_LSECSS_EXTI_CLEAR_FLAG	**Description:** • Clear the RCC LSE CSS EXTI flag. **Return value:** • None.
__HAL_RCC_LSECSS_EXTI_GENERATE_ SWIT	**Description:** • Generate a Software interrupt on the RCC LSE CSS EXTI line.

Return value:

- None.

__HAL_RCC_CRS_ENABLE_IT

Description:

- Enable the specified CRS interrupts.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

__HAL_RCC_CRS_DISABLE_IT

Description:

- Disable the specified CRS interrupts.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

__HAL_RCC_CRS_GET_IT_SOURCE

Description:

- Check whether the CRS interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt source to check. This parameter can be one of the following values:

- RCC_CRS_IT_SYNCOK SYNC event OK interrupt
- RCC_CRS_IT_SYNCWARN SYNC warning interrupt
- RCC_CRS_IT_ERR Synchronization or trimming error interrupt
- RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

RCC_CRS_IT_ERROR_MASK

Description:

- Clear the CRS interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt
  - RCC_CRS_IT_TRIMOVF Trimming overflow or underflow interrupt
  - RCC_CRS_IT_SYNCERR SYNC error interrupt
  - RCC_CRS_IT_SYNCMISS SYNC missed interrupt

__HAL_RCC_CRS_CLEAR_IT

__HAL_RCC_CRS_GET_FLAG

Description:

- Check whether the specified CRS flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_CRS_FLAG_SYNCOK SYNC event OK
  - RCC_CRS_FLAG_SYNCWARN SYNC warning
  - RCC_CRS_FLAG_ERR Error
  - RCC_CRS_FLAG_ESYNC

Expected SYNC
- RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
- RCC_CRS_FLAG_SYNCERR SYNC error
- RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- The: new state of _FLAG_ (TRUE or FALSE).

RCC_CRS_FLAG_ERROR_MASK

Description:

- Clear the CRS specified FLAG.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - RCC_CRS_FLAG_SYNCOK SYNC event OK
  - RCC_CRS_FLAG_SYNCWARN SYNC warning
  - RCC_CRS_FLAG_ERR Error
  - RCC_CRS_FLAG_ESYNC Expected SYNC
  - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
  - RCC_CRS_FLAG_SYNCERR SYNC error
  - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- None

Notes:

- RCC_CRS_FLAG_ERR clears RCC_CRS_FLAG_TRIMOVF, RCC_CRS_FLAG_SYNCERR, RCC_CRS_FLAG_SYNCMISS and consequently RCC_CRS_FLAG_ERR

__HAL_RCC_CRS_CLEAR_FLAG

I2C1 Clock Source

RCC_I2C1CLKSOURCE_PCLK1

RCC_I2C1CLKSOURCE_SYSCLK

RCC_I2C1CLKSOURCE_HSI

I2C2 Clock Source

RCC_I2C2CLKSOURCE_PCLK1

RCC_I2C2CLKSOURCE_SYSCLK

RCC_I2C2CLKSOURCE_HSI

I2C3 Clock Source

RCC_I2C3CLKSOURCE_PCLK1

RCC_I2C3CLKSOURCE_SYSCLK

RCC_I2C3CLKSOURCE_HSI

I2C4 Clock Source

RCC_I2C4CLKSOURCE_PCLK1

RCC_I2C4CLKSOURCE_SYSCLK

RCC_I2C4CLKSOURCE_HSI

LPTIM1 Clock Source

RCC_LPTIM1CLKSOURCE_PCLK1

RCC_LPTIM1CLKSOURCE_LSI

RCC_LPTIM1CLKSOURCE_HSI

RCC_LPTIM1CLKSOURCE_LSE

LPTIM2 Clock Source

RCC_LPTIM2CLKSOURCE_PCLK1

RCC_LPTIM2CLKSOURCE_LSI

RCC_LPTIM2CLKSOURCE_HSI

RCC_LPTIM2CLKSOURCE_LSE

LPUART1 Clock Source

RCC_LPUART1CLKSOURCE_PCLK1

RCC_LPUART1CLKSOURCE_SYSCLK

RCC_LPUART1CLKSOURCE_HSI

RCC_LPUART1CLKSOURCE_LSE

Low Speed Clock Source

RCC_LSCOSOURCE_LSI	LSI selection for low speed clock output
RCC_LSCOSOURCE_LSE	LSE selection for low speed clock output

LTDC Clock Source

RCC_LTDCCLKSOURCE_PLLSAI2_DIV2

RCC_LTDCCLKSOURCE_PLLSAI2_DIV4

RCC_LTDCCLKSOURCE_PLLSAI2_DIV8

RCC_LTDCCLKSOURCE_PLLSAI2_DIV16

OctoSPI Clock Source

RCC_OSPICLKSOURCE_SYSCLK

RCC_OSPICLKSOURCE_MSI

RCC_OSPICLKSOURCE_PLL

Periph Clock Selection

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_USART2

RCC_PERIPHCLK_USART3

RCC_PERIPHCLK_UART4

RCC_PERIPHCLK_UART5

RCC_PERIPHCLK_LPUART1

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2C2

RCC_PERIPHCLK_I2C3

RCC_PERIPHCLK_LPTIM1

RCC_PERIPHCLK_LPTIM2

RCC_PERIPHCLK_SAI1

RCC_PERIPHCLK_SAI2

RCC_PERIPHCLK_USB

RCC_PERIPHCLK_ADC

RCC_PERIPHCLK_DFSDM1

RCC_PERIPHCLK_DFSDM1AUDIO

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_RNG

RCC_PERIPHCLK_SDMMC1

RCC_PERIPHCLK_I2C4

RCC_PERIPHCLK_LTDC

RCC_PERIPHCLK_DSI

RCC_PERIPHCLK_OSPI

RNG Clock Source

RCC_RNGCLKSOURCE_HSI48

RCC_RNGCLKSOURCE_PLLSAI1

RCC_RNGCLKSOURCE_PLL

RCC_RNGCLKSOURCE_MSI

SAI1 Clock Source

RCC_SAI1CLKSOURCE_PLLSAI1

RCC_SAI1CLKSOURCE_PLLSAI2

RCC_SAI1CLKSOURCE_PLL

RCC_SAI1CLKSOURCE_PIN

RCC_SAI1CLKSOURCE_HSI

SAI2 Clock Source

RCC_SAI2CLKSOURCE_PLLSAI1

RCC_SAI2CLKSOURCE_PLLSAI2

RCC_SAI2CLKSOURCE_PLL

RCC_SAI2CLKSOURCE_PIN

RCC_SAI2CLKSOURCE_HSI

SDMMC1 Clock Source

RCC_SDMMC1CLKSOURCE_HSI48

RCC_SDMMC1CLKSOURCE_PLLSAI1

RCC_SDMMC1CLKSOURCE_PLL

RCC_SDMMC1CLKSOURCE_MSI

UART4 Clock Source

RCC_UART4CLKSOURCE_PCLK1

RCC_UART4CLKSOURCE_SYSCLK

RCC_UART4CLKSOURCE_HSI

RCC_UART4CLKSOURCE_LSE

UART5 Clock Source

RCC_UART5CLKSOURCE_PCLK1

RCC_UART5CLKSOURCE_SYSCLK

RCC_UART5CLKSOURCE_HSI

RCC_UART5CLKSOURCE_LSE

USART1 Clock Source

RCC_USART1CLKSOURCE_PCLK2

RCC_USART1CLKSOURCE_SYSCLK

RCC_USART1CLKSOURCE_HSI

RCC_USART1CLKSOURCE_LSE

USART2 Clock Source

RCC_USART2CLKSOURCE_PCLK1

RCC_USART2CLKSOURCE_SYSCLK

RCC_USART2CLKSOURCE_HSI

RCC_USART2CLKSOURCE_LSE

USART3 Clock Source

RCC_USART3CLKSOURCE_PCLK1

RCC_USART3CLKSOURCE_SYSCLK

RCC_USART3CLKSOURCE_HSI

RCC_USART3CLKSOURCE_LSE

USB Clock Source

RCC_USBCLKSOURCE_HSI48

RCC_USBCLKSOURCE_PLLSAI1

RCC_USBCLKSOURCE_PLL

RCC_USBCLKSOURCE_MSI

53 HAL RNG Generic Driver

53.1 RNG Firmware driver registers structures

53.1.1 RNG_InitTypeDef

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t RNG_InitTypeDef::ClockErrorDetection***
  Clock error detection

53.1.2 NG_HandleTypeDef

Data Fields

- ***RNG_TypeDef * Instance***
- ***RNG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RNG_StateTypeDef State***
- ***uint32_t RandomNumber***

Field Documentation

- ***RNG_TypeDef* RNG_HandleTypeDef::Instance***
  Register base address
- ***RNG_InitTypeDef RNG_HandleTypeDef::Init***
  RNG configuration parameters
- ***HAL_LockTypeDef RNG_HandleTypeDef::Lock***
  RNG locking object
- ***__IO HAL_RNG_StateTypeDef RNG_HandleTypeDef::State***
  RNG communication state
- ***uint32_t RNG_HandleTypeDef::RandomNumber***
  Last Generated RNG Data

53.2 RNG Firmware driver API description

53.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using __HAL_RCC_RNG_CLK_ENABLE() macro in HAL_RNG_MspInit().
2. Activate the RNG peripheral using HAL_RNG_Init() function.
3. Wait until the 32-bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using HAL_RNG_GenerateRandomNumber() function.

53.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the RNG according to the specified parameters in the RNG_InitTypeDef and create the associated handle

- DeInitialize the RNG peripheral
- Initialize the RNG MSP (MCU Specific Package)
- DeInitialize the RNG MSP

This section contains the following APIs:

- *HAL_RNG_Init()*
- *HAL_RNG_DeInit()*
- *HAL_RNG_MspInit()*
- *HAL_RNG_MspDeInit()*

53.2.3 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- Handle RNG interrupt request

This section contains the following APIs:

- *HAL_RNG_GenerateRandomNumber()*
- *HAL_RNG_GenerateRandomNumber_IT()*
- *HAL_RNG_IRQHandler()*
- *HAL_RNG_GetRandomNumber()*
- *HAL_RNG_GetRandomNumber_IT()*
- *HAL_RNG_ReadLastRandomNumber()*
- *HAL_RNG_ReadyDataCallback()*
- *HAL_RNG_ErrorCallback()*

53.2.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_RNG_GetState()*

53.2.5 Detailed description of functions

HAL_RNG_Init

Function name	**HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)**
Function description	Initialize the RNG peripheral and initialize the associated handle.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **HAL:** status

HAL_RNG_DeInit

Function name	**HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)**
Function description	DeInitialize the RNG peripheral.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **HAL:** status

HAL_RNG_MspInit

Function name	**void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)**
Function description	Initialize the RNG MSP.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **None:**

HAL_RNG_MspDeInit

Function name	**void HAL_RNG_MspDeInit (RNG_HandleTypeDef * hrng)**
Function description	DeInitialize the RNG MSP.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **None:**

HAL_RNG_GetRandomNumber

Function name	**uint32_t HAL_RNG_GetRandomNumber (RNG_HandleTypeDef * hrng)**
Function description	Return generated random number in polling mode (Obsolete).
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	• **random:** value
Notes	• Use HAL_RNG_GenerateRandomNumber() API instead.

HAL_RNG_GetRandomNumber_IT

Function name	**uint32_t HAL_RNG_GetRandomNumber_IT (RNG_HandleTypeDef * hrng)**
Function description	Return a 32-bit random number with interrupt enabled (Obsolete).
Parameters	• **hrng:** RNG handle
Return values	• **32-bit:** random number
Notes	• Use HAL_RNG_GenerateRandomNumber_IT() API instead.

HAL_RNG_GenerateRandomNumber

Function name	**HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random32bit)**
Function description	Generate a 32-bit random number.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure. • **random32bit:** pointer to generated random number variable if successful.
Return values	• **HAL:** status
Notes	• Each time the random number data is read the RNG_FLAG_DRDY flag is automatically cleared.

HAL_RNG_GenerateRandomNumber_IT

Function name	**HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber_IT (RNG_HandleTypeDef * hrng)**
Function description	Generate a 32-bit random number in interrupt mode.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **HAL:** status

HAL_RNG_ReadLastRandomNumber

Function name	**uint32_t HAL_RNG_ReadLastRandomNumber (RNG_HandleTypeDef * hrng)**
Function description	Read latest generated random number.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **random:** value

HAL_RNG_IRQHandler

Function name	**void HAL_RNG_IRQHandler (RNG_HandleTypeDef * hrng)**
Function description	Handle RNG interrupt request.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **None:**
Notes	• In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using __HAL_RNG_CLEAR_IT(). The clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can be used. • In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using __HAL_RNG_CLEAR_IT(), then disable and enable the RNG peripheral to reinitialize and restart the RNG. • User-written HAL_RNG_ErrorCallback() API is called once whether SEIS or CEIS are set.

HAL_RNG_ErrorCallback

Function name	**void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * hrng)**
Function description	RNG error callback.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **None:**

HAL_RNG_ReadyDataCallback

Function name	**void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)**
Function description	Data Ready callback in non-blocking mode.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure. • **random32bit:** generated random value
Return values	• **None:**

HAL_RNG_GetState

Function name	**HAL_RNG_StateTypeDef HAL_RNG_GetState (RNG_HandleTypeDef * hrng)**
Function description	Return the RNG handle state.
Parameters	• **hrng:** pointer to a RNG_HandleTypeDef structure.
Return values	• **HAL:** state

53.3 RNG Firmware driver defines

53.3.1 RNG

RNG Clock Error Detection

RNG_CED_ENABLE	Clock error detection enabled
RNG_CED_DISABLE	Clock error detection disabled

RNG Exported Macros

__HAL_RNG_RESET_HANDLE_STATE

Description:

- Reset RNG handle state.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_ENABLE

Description:

- Enable the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE

Description:

- Disable the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_FLAG

Description:

- Check whether the specified RNG flag is set or not.

Parameters:

- __HANDLE__: RNG Handle
- __FLAG__: RNG flag This parameter can be one of the following values:
  – RNG_FLAG_DRDY: Data ready
  – RNG_FLAG_CECS: Clock error current status
  – RNG_FLAG_SECS: Seed error current status

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_RNG_CLEAR_FLAG

Description:

- Clear the selected RNG flag status.

Parameters:

- __HANDLE__: RNG handle
- __FLAG__: RNG flag to clear

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment, flags RNG_FLAG_DRDY, RNG_FLAG_CECS and RNG_FLAG_SECS are read-only.

__HAL_RNG_ENABLE_IT

Description:

- Enable the RNG interrupt.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE_IT

Description:

- Disable the RNG interrupt.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_IT

Description:

- Check whether the specified RNG interrupt has occurred or not.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
  - RNG_IT_DRDY: Data ready interrupt
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_RNG_CLEAR_IT

Description:

- Clear the RNG interrupt status flags.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- None

Notes:

- RNG_IT_DRDY flag is read-only, reading RNG_DR register automatically clears RNG_IT_DRDY.

RNG Flags Definition

RNG_FLAG_DRDY	Data ready
RNG_FLAG_CECS	Clock error current status
RNG_FLAG_SECS	Seed error current status

RNG Interrupts Definition

RNG_IT_DRDY	Data Ready interrupt
RNG_IT_CEI	Clock error interrupt
RNG_IT_SEI	Seed error interrupt

54 HAL RTC Generic Driver

54.1 RTC Firmware driver registers structures

54.1.1 RTC_InitTypeDef

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrediv***
- ***uint32_t SynchPrediv***
- ***uint32_t OutPut***
- ***uint32_t OutPutRemap***
- ***uint32_t OutPutPolarity***
- ***uint32_t OutPutType***

Field Documentation

- ***uint32_t RTC_InitTypeDef::HourFormat***
  Specifies the RTC Hour Format. This parameter can be a value of ***RTC_Hour_Formats***
- ***uint32_t RTC_InitTypeDef::AsynchPrediv***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F
- ***uint32_t RTC_InitTypeDef::SynchPrediv***
  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFF
- ***uint32_t RTC_InitTypeDef::OutPut***
  Specifies which signal will be routed to the RTC output. This parameter can be a value of ***RTCEx_Output_selection_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutRemap***
  Specifies the remap for RTC output. This parameter can be a value of ***RTC_Output_ALARM_OUT_Remap***
- ***uint32_t RTC_InitTypeDef::OutPutPolarity***
  Specifies the polarity of the output signal. This parameter can be a value of ***RTC_Output_Polarity_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutType***
  Specifies the RTC Output Pin mode. This parameter can be a value of ***RTC_Output_Type_ALARM_OUT***

54.1.2 RTC_TimeTypeDef

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***
- ***uint8_t TimeFormat***
- ***uint32_t SubSeconds***
- ***uint32_t SecondFraction***
- ***uint32_t DayLightSaving***
- ***uint32_t StoreOperation***

Field Documentation

- ***uint8_t RTC_TimeTypeDef::Hours***
  Specifies the RTC Time Hour. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the RTC_HourFormat_12 is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the RTC_HourFormat_24 is selected
- ***uint8_t RTC_TimeTypeDef::Minutes***
  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::TimeFormat***
  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_AM_PM_Definitions***
- ***uint32_t RTC_TimeTypeDef::SubSeconds***
  Specifies the RTC_SSR RTC Sub Second register content. This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity
- ***uint32_t RTC_TimeTypeDef::SecondFraction***
  Specifies the range or granularity of Sub Second register content corresponding to Synchronous pre-scaler factor value (PREDIV_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL_RTC_GetTime function
- ***uint32_t RTC_TimeTypeDef::DayLightSaving***
  Specifies RTC_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of ***RTC_DayLightSaving_Definitions***
- ***uint32_t RTC_TimeTypeDef::StoreOperation***
  Specifies RTC_StoreOperation value to be written in the BCK bit in CR register to store the operation. This parameter can be a value of ***RTC_StoreOperation_Definitions***

54.1.3 RTC_DateTypeDef

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Date***
- ***uint8_t Year***

Field Documentation

- ***uint8_t RTC_DateTypeDef::WeekDay***
  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint8_t RTC_DateTypeDef::Month***
  Specifies the RTC Date Month (in BCD format). This parameter can be a value of ***RTC_Month_Date_Definitions***
- ***uint8_t RTC_DateTypeDef::Date***
  Specifies the RTC Date. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***uint8_t RTC_DateTypeDef::Year***
  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99

54.1.4 RTC_AlarmTypeDef

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmSubSecondMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***
- ***uint32_t Alarm***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members
- ***uint32_t RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_AlarmMask_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmSubSecondMask***
  Specifies the RTC Alarm SubSeconds Masks. This parameter can be a value of ***RTC_Alarm_Sub_Seconds_Masks_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of ***RTC_AlarmDateWeekDay_Definitions***
- ***uint8_t RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint32_t RTC_AlarmTypeDef::Alarm***
  Specifies the alarm . This parameter can be a value of ***RTC_Alarms_Definitions***

54.1.5 RTC_HandleTypeDef

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RTCStateTypeDef State***

Field Documentation

- ***RTC_TypeDef* RTC_HandleTypeDef::Instance***
  Register base address
- ***RTC_InitTypeDef RTC_HandleTypeDef::Init***
  RTC required parameters
- ***HAL_LockTypeDef RTC_HandleTypeDef::Lock***
  RTC locking object
- ***__IO HAL_RTCStateTypeDef RTC_HandleTypeDef::State***
  Time communication state

54.2 RTC Firmware driver API description

54.2.1 RTC Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

54.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values. A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.
3. Tamper detection event resets all data backup registers.

54.2.3 Backup Domain Access

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

1. Call the function HAL_RCCEx_PeriphCLKConfig with RCC_PERIPHCLK_RTC for PeriphClockSelection and select RTCClockSelection (LSE, LSI or HSEdiv32)
2. Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() macro.

54.2.4 How to use RTC Driver

1. Enable the RTC domain access (see description in the section above).
2. Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

Time and Date configuration

1. To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
2. To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.

Alarm configuration

1. To configure the RTC Alarm use the HAL_RTC_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL_RTC_SetAlarm_IT() function.
2. To read the RTC Alarm, use the HAL_RTC_GetAlarm() function.

54.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and Standby modes is possible only when the RTC clock source is LSE or LSI.

54.2.6 Initialization and de-initialization functions

This section provide functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
   - A 7-bit asynchronous prescaler and a 15-bit synchronous prescaler.
   - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers. The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- ***HAL_RTC_Init()***
- ***HAL_RTC_DeInit()***
- ***HAL_RTC_MspInit()***
- ***HAL_RTC_MspDeInit()***

54.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

- ***HAL_RTC_SetTime()***
- ***HAL_RTC_GetTime()***
- ***HAL_RTC_SetDate()***
- ***HAL_RTC_GetDate()***

54.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:

- ***HAL_RTC_SetAlarm()***
- ***HAL_RTC_SetAlarm_IT()***
- ***HAL_RTC_DeactivateAlarm()***
- ***HAL_RTC_GetAlarm()***
- ***HAL_RTC_AlarmIRQHandler()***
- ***HAL_RTC_AlarmAEventCallback()***
- ***HAL_RTC_PollForAlarmAEvent()***

54.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

- *HAL_RTC_WaitForSynchro()*

54.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- *HAL_RTC_GetState()*

54.2.11 Detailed description of functions

HAL_RTC_Init

Function name	**HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)**
Function description	Initialize the RTC according to the specified parameters in the RTC_InitTypeDef structure and initialize the associated handle.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTC_DeInit

Function name	**HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)**
Function description	DeInitialize the RTC peripheral.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status
Notes	• This function doesn't reset the RTC Backup Data registers.

HAL_RTC_MspInit

Function name	**void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)**
Function description	Initialize the RTC MSP.
Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTC_MspDeInit

Function name	**void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)**
Function description	DeInitialize the RTC MSP.

Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTC_SetTime

Function name	**HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)**
Function description	Set RTC current time.
Parameters	• **hrtc:** RTC handle • **sTime:** Pointer to Time structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status

HAL_RTC_GetTime

Function name	**HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)**
Function description	Get RTC current time.
Parameters	• **hrtc:** RTC handle • **sTime:** Pointer to Time structure with Hours, Minutes and Seconds fields returned with input format (BIN or BCD), also SubSeconds field returning the RTC_SSR register content and SecondFraction field the Synchronous pre-scaler factor to be used for second fraction ratio computation. • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status
Notes	• You can use SubSeconds and SecondFraction (sTime structure fields returned) to convert SubSeconds value in second fraction ratio with time unit following generic formula: Second fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS • You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read to ensure consistency between the time and date values.

HAL_RTC_SetDate

Function name	**HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef**

	*** hrtc, RTC_DateTypeDef * sDate, uint32_t Format)**
Function description	Set RTC current date.
Parameters	• **hrtc:** RTC handle • **sDate:** Pointer to date structure • **Format:** specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status

HAL_RTC_GetDate

Function name	**HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)**
Function description	Get RTC current date.
Parameters	• **hrtc:** RTC handle • **sDate:** Pointer to Date structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status
Notes	• You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

HAL_RTC_SetAlarm

Function name	**HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)**
Function description	Set the specified RTC Alarm.
Parameters	• **hrtc:** RTC handle • **sAlarm:** Pointer to Alarm structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status

HAL_RTC_SetAlarm_IT

Function name	**HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)**
Function description	Set the specified RTC Alarm with Interrupt.

Parameters	• **hrtc:** RTC handle • **sAlarm:** Pointer to Alarm structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status
Notes	• The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()). • The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

HAL_RTC_DeactivateAlarm

Function name	**HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)**
Function description	Deactivate the specified RTC Alarm.
Parameters	• **hrtc:** RTC handle • **Alarm:** Specifies the Alarm. This parameter can be one of the following values: – RTC_ALARM_A: AlarmA – RTC_ALARM_B: AlarmB
Return values	• **HAL:** status

HAL_RTC_GetAlarm

Function name	**HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)**
Function description	Get the RTC Alarm value and masks.
Parameters	• **hrtc:** RTC handle • **sAlarm:** Pointer to Date structure • **Alarm:** Specifies the Alarm. This parameter can be one of the following values: – RTC_ALARM_A: AlarmA – RTC_ALARM_B: AlarmB • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status

HAL_RTC_AlarmIRQHandler

Function name	**void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)**
Function description	Handle Alarm interrupt request.
Parameters	• **hrtc:** RTC handle

Return values	• **None:**

HAL_RTC_PollForAlarmAEvent

Function name	**HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle AlarmA Polling request.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_RTC_AlarmAEventCallback

Function name	**void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Alarm A callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTC_WaitForSynchro

Function name	**HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)**
Function description	Wait until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status
Notes	• The RTC Resynchronization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function. • To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

HAL_RTC_GetState

Function name	**HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)**
Function description	Return the RTC handle state.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** state

RTC_EnterInitMode

Function name	**HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)**
Function description	Enter the RTC Initialization mode.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status
Notes	• The RTC Initialization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.

RTC_ByteToBcd2

Function name	**uint8_t RTC_ByteToBcd2 (uint8_t Value)**
Function description	Convert a 2 digit decimal to BCD format.
Parameters	• **Value:** Byte to be converted
Return values	• **Converted:** byte

RTC_Bcd2ToByte

Function name	**uint8_t RTC_Bcd2ToByte (uint8_t Value)**
Function description	Convert from 2 digit BCD to Binary.
Parameters	• **Value:** BCD value to be converted
Return values	• **Converted:** word

54.3 RTC Firmware driver defines

54.3.1 RTC

RTC Alarm Date WeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE

RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

RTC Alarm Mask Definitions

RTC_ALARMMASK_NONE

RTC_ALARMMASK_DATEWEEKDAY

RTC_ALARMMASK_HOURS

RTC_ALARMMASK_MINUTES

RTC_ALARMMASK_SECONDS

RTC_ALARMMASK_ALL

RTC Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

RTC Alarm Sub Seconds Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL	All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm
RTC_ALARMSUBSECONDMASK_SS14_1	SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.
RTC_ALARMSUBSECONDMASK_SS14_2	SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_3	SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_4	SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_5	SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_6	SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_7	SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_8	SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_9	SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_10	SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_11	SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_12	SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_13	SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared
RTC_ALARMSUBSECONDMASK_SS14	SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared
RTC_ALARMSUBSECONDMASK_NONE	SS[14:0] are compared and must match to activate alarm.

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC DayLight Saving Definitions

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

RTC Exported Macros

__HAL_RTC_RESET_HANDLE_STATE

Description:

- Reset RTC handle state.

	Parameters: • __HANDLE__: RTC handle. **Return value:** • None
__HAL_RTC_WRITEPROTECTION_DISABLE	**Description:** • Disable the write protection for RTC registers. **Parameters:** • __HANDLE__: specifies the RTC handle. **Return value:** • None
__HAL_RTC_WRITEPROTECTION_ENABLE	**Description:** • Enable the write protection for RTC registers. **Parameters:** • __HANDLE__: specifies the RTC handle. **Return value:** • None
__HAL_RTC_ALARMA_ENABLE	**Description:** • Enable the RTC ALARMA peripheral. **Parameters:** • __HANDLE__: specifies the RTC handle. **Return value:** • None
__HAL_RTC_ALARMA_DISABLE	**Description:** • Disable the RTC ALARMA peripheral. **Parameters:** • __HANDLE__: specifies the RTC handle. **Return value:** • None
__HAL_RTC_ALARMB_ENABLE	**Description:** • Enable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_DISABLE

Description:

- Disable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARM_ENABLE_IT

Description:

- Enable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  – RTC_IT_ALRA: Alarm A interrupt
  – RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_DISABLE_IT

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  – RTC_IT_ALRA: Alarm A interrupt
  – RTC_IT_ALRB: Alarm B

interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT

Description:

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA: Alarm A interrupt
  - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_FLAG

Description:

- Get the selected RTC Alarm's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to check. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF
  - RTC_FLAG_ALRAWF
  - RTC_FLAG_ALRBWF

Return value:

- None

__HAL_RTC_ALARM_CLEAR_FLAG

Description:

- Clear the RTC Alarm's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  - RTC_FLAG_ALRAF

– RTC_FLAG_ALRBF

Return value:

- None

__HAL_RTC_ALARM_GET_IT_SOURCE

Description:

- Check whether the specified RTC Alarm interrupt is enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  – RTC_IT_ALRA: Alarm A interrupt
  – RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable falling edge trigger on

the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE

Description:

- Enable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE

Description:

- Disable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_GET_FLAG

Description:

- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

__HAL_RTC_ALARM_EXTI_CLEAR_FLAG

Description:

- Clear the RTC Alarm associated Exti line flag.

Return value:

- None

__HAL_RTC_ALARM_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on RTC Alarm associated Exti line.

Return value:

- None

RTC Flags Definitions

RTC_FLAG_RECALPF

RTC_FLAG_TAMP3F

RTC_FLAG_TAMP2F

RTC_FLAG_TAMP1F

RTC_FLAG_TSOVF

RTC_FLAG_TSF

RTC_FLAG_ITSF

RTC_FLAG_WUTF

RTC_FLAG_ALRBF

RTC_FLAG_ALRAF

RTC_FLAG_INITF

RTC_FLAG_RSF

RTC_FLAG_INITS

RTC_FLAG_SHPF

RTC_FLAG_WUTWF

RTC_FLAG_ALRBWF

RTC_FLAG_ALRAWF

RTC Hour Formats

RTC_HOURFORMAT_24

RTC_HOURFORMAT_12

RTC Input Parameter Format Definitions

RTC_FORMAT_BIN

RTC_FORMAT_BCD

RTC Interrupts Definitions

RTC_IT_TS	Enable Timestamp Interrupt
RTC_IT_WUT	Enable Wakeup timer Interrupt
RTC_IT_ALRA	Enable Alarm A Interrupt

RTC_IT_ALRB	Enable Alarm B Interrupt
RTC_IT_TAMP	Enable all Tamper Interrupt
RTC_IT_TAMP1	Enable Tamper 1 Interrupt
RTC_IT_TAMP2	Enable Tamper 2 Interrupt
RTC_IT_TAMP3	Enable Tamper 3 Interrupt

RTC Private macros to check input parameters

IS_RTC_HOUR_FORMAT

IS_RTC_OUTPUT_POL

IS_RTC_OUTPUT_TYPE

IS_RTC_OUTPUT_REMAP

IS_RTC_HOURFORMAT12

IS_RTC_DAYLIGHT_SAVING

IS_RTC_STORE_OPERATION

IS_RTC_FORMAT

IS_RTC_YEAR

IS_RTC_MONTH

IS_RTC_DATE

IS_RTC_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_DATE

IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_SEL

IS_RTC_ALARM_MASK

IS_RTC_ALARM

IS_RTC_ALARM_SUB_SECOND_VALUE

IS_RTC_ALARM_SUB_SECOND_MASK

IS_RTC_ASYNCH_PREDIV

IS_RTC_SYNCH_PREDIV

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

RTC Month Date Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

RTC Output ALARM OUT Remap

RTC_OUTPUT_REMAP_NONE

RTC_OUTPUT_REMAP_POS1

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_OPENDRAIN

RTC_OUTPUT_TYPE_PUSHPULL

RTC Store Operation Definitions

RTC_STOREOPERATION_RESET

RTC_STOREOPERATION_SET

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

55 HAL RTC Extension Driver

55.1 RTCEx Firmware driver registers structures

55.1.1 RTC_TamperTypeDef

Data Fields

- ***uint32_t Tamper***
- ***uint32_t Interrupt***
- ***uint32_t Trigger***
- ***uint32_t NoErase***
- ***uint32_t MaskFlag***
- ***uint32_t Filter***
- ***uint32_t SamplingFrequency***
- ***uint32_t PrechargeDuration***
- ***uint32_t TamperPullUp***
- ***uint32_t TimeStampOnTamperDetection***

Field Documentation

- ***uint32_t RTC_TamperTypeDef::Tamper***
  Specifies the Tamper Pin. This parameter can be a value of
  RTCEx_Tamper_Pins_Definitions
- ***uint32_t RTC_TamperTypeDef::Interrupt***
  Specifies the Tamper Interrupt. This parameter can be a value of
  RTCEx_Tamper_Interrupt_Definitions
- ***uint32_t RTC_TamperTypeDef::Trigger***
  Specifies the Tamper Trigger. This parameter can be a value of
  RTCEx_Tamper_Trigger_Definitions
- ***uint32_t RTC_TamperTypeDef::NoErase***
  Specifies the Tamper no erase mode. This parameter can be a value of
  RTCEx_Tamper_EraseBackUp_Definitions
- ***uint32_t RTC_TamperTypeDef::MaskFlag***
  Specifies the Tamper Flag masking. This parameter can be a value of
  RTCEx_Tamper_MaskFlag_Definitions
- ***uint32_t RTC_TamperTypeDef::Filter***
  Specifies the RTC Filter Tamper. This parameter can be a value of
  RTCEx_Tamper_Filter_Definitions
- ***uint32_t RTC_TamperTypeDef::SamplingFrequency***
  Specifies the sampling frequency. This parameter can be a value of
  RTCEx_Tamper_Sampling_Frequencies_Definitions
- ***uint32_t RTC_TamperTypeDef::PrechargeDuration***
  Specifies the Precharge Duration . This parameter can be a value of
  RTCEx_Tamper_Pin_Precharge_Duration_Definitions
- ***uint32_t RTC_TamperTypeDef::TamperPullUp***
  Specifies the Tamper PullUp . This parameter can be a value of
  RTCEx_Tamper_Pull_UP_Definitions
- ***uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of
  RTCEx_Tamper_TimeStampOnTamperDetection_Definitions

55.2 RTCEx Firmware driver API description

55.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTCEx_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer with interrupt mode using the HAL_RTCEx_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTCEx_GetWakeUpTimer() function.

Outputs configuration

The RTC has 2 different outputs:

- RTC_ALARM: this output is used to manage the RTC Alarm A, Alarm B and WaKeUp signals. To output the selected RTC signal, use the HAL_RTC_Init() function.
- RTC_CALIB: this output is 512Hz signal or 1Hz. To enable the RTC_CALIB, use the HAL_RTCEx_SetCalibrationOutPut() function.
- Two pins can be used as RTC_ALARM or RTC_CALIB (PC13, PB2) managed on the RTC_OR register.
- When the RTC_CALIB or RTC_ALARM output is selected, the RTC_OUT pin is automatically configured in output alternate function.

Smooth digital Calibration configuration

- Configure the RTC Original Digital Calibration Value and the corresponding calibration cycle period (32s,16s and 8s) using the HAL_RTCEx_SetSmoothCalib() function.

TimeStamp configuration

- Enable the RTC TimeStamp using the HAL_RTCEx_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTCEx_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Internal TimeStamp configuration

- Enable the RTC internal TimeStamp using the HAL_RTCEx_SetInternalTimeStamp() function. User has to check internal timestamp occurrence using __HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP using the HAL_RTCEx_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.

- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the RTC_TAMPCR register.

Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL_RTCEx_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL_RTCEx_BKUPRead() function.

55.2.2 RTC TimeStamp and Tamper functions

This section provide functions allowing to configure TimeStamp feature

This section contains the following APIs:

- *HAL_RTCEx_SetTimeStamp()*
- *HAL_RTCEx_SetTimeStamp_IT()*
- *HAL_RTCEx_DeactivateTimeStamp()*
- *HAL_RTCEx_SetInternalTimeStamp()*
- *HAL_RTCEx_DeactivateInternalTimeStamp()*
- *HAL_RTCEx_GetTimeStamp()*
- *HAL_RTCEx_SetTamper()*
- *HAL_RTCEx_SetTamper_IT()*
- *HAL_RTCEx_DeactivateTamper()*
- *HAL_RTCEx_TamperTimeStampIRQHandler()*
- *HAL_RTCEx_TimeStampEventCallback()*
- *HAL_RTCEx_Tamper1EventCallback()*
- *HAL_RTCEx_Tamper2EventCallback()*
- *HAL_RTCEx_Tamper3EventCallback()*
- *HAL_RTCEx_PollForTimeStampEvent()*
- *HAL_RTCEx_PollForTamper1Event()*
- *HAL_RTCEx_PollForTamper2Event()*
- *HAL_RTCEx_PollForTamper3Event()*

55.2.3 RTC Wake-up functions

This section provide functions allowing to configure Wake-up feature

This section contains the following APIs:

- *HAL_RTCEx_SetWakeUpTimer()*
- *HAL_RTCEx_SetWakeUpTimer_IT()*
- *HAL_RTCEx_DeactivateWakeUpTimer()*
- *HAL_RTCEx_GetWakeUpTimer()*
- *HAL_RTCEx_WakeUpTimerIRQHandler()*
- *HAL_RTCEx_WakeUpTimerEventCallback()*
- *HAL_RTCEx_PollForWakeUpTimerEvent()*

55.2.4 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.

- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.

This section contains the following APIs:

- ***HAL_RTCEx_BKUPWrite()***
- ***HAL_RTCEx_BKUPRead()***
- ***HAL_RTCEx_SetSmoothCalib()***
- ***HAL_RTCEx_SetSynchroShift()***
- ***HAL_RTCEx_SetCalibrationOutPut()***
- ***HAL_RTCEx_DeactivateCalibrationOutPut()***
- ***HAL_RTCEx_SetRefClock()***
- ***HAL_RTCEx_DeactivateRefClock()***
- ***HAL_RTCEx_EnableBypassShadow()***
- ***HAL_RTCEx_DisableBypassShadow()***

55.2.5 Extended features functions

This section provides functions allowing to:

- RTC Alarm B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- ***HAL_RTCEx_AlarmBEventCallback()***
- ***HAL_RTCEx_PollForAlarmBEvent()***

55.2.6 Detailed description of functions

HAL_RTCEx_SetTimeStamp

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)**
Function description	Set TimeStamp.
Parameters	• **hrtc:** RTC handle • **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: – RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin. – RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin. • **RTC_TimeStampPin:** specifies the RTC TimeStamp Pin. This parameter can be one of the following values: – RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values	• **HAL:** status
Notes	• This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_SetTimeStamp_IT

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)**
Function description	Set TimeStamp with Interrupt.
Parameters	• **hrtc:** RTC handle • **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: – RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin. – RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin. • **RTC_TimeStampPin:** Specifies the RTC TimeStamp Pin. This parameter can be one of the following values: – RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.
Return values	• **HAL:** status
Notes	• This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_DeactivateTimeStamp

Function name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)**
Function description	Deactivate TimeStamp.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTCEx_SetInternalTimeStamp

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp (RTC_HandleTypeDef * hrtc)**
Function description	Set Internal TimeStamp.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• **HAL:** status
Notes	• This API must be called before enabling the internal TimeStamp feature.

HAL_RTCEx_DeactivateInternalTimeStamp

Function name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp (RTC_HandleTypeDef * hrtc)**
Function description	Deactivate Internal TimeStamp.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• **HAL:** status

HAL_RTCEx_GetTimeStamp

Function name	**HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)**
Function description	Get the RTC TimeStamp value.
Parameters	• **hrtc:** RTC handle • **sTimeStamp:** Pointer to Time structure • **sTimeStampDate:** Pointer to Date structure • **Format:** specifies the format of the entered parameters. This parameter can be one of the following values: – RTC_FORMAT_BIN: Binary data format – RTC_FORMAT_BCD: BCD data format
Return values	• **HAL:** status

HAL_RTCEx_SetTamper

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)**
Function description	Set Tamper.
Parameters	• **hrtc:** RTC handle • **sTamper:** Pointer to Tamper Structure.
Return values	• **HAL:** status
Notes	• By calling this API we disable the tamper interrupt for all tampers.

HAL_RTCEx_SetTamper_IT

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)**
Function description	Set Tamper with interrupt.
Parameters	• **hrtc:** RTC handle • **sTamper:** Pointer to RTC Tamper.
Return values	• **HAL:** status
Notes	• By calling this API we force the tamper interrupt for all

tampers.

HAL_RTCEx_DeactivateTamper

Function name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)**
Function description	Deactivate Tamper.
Parameters	• **hrtc:** RTC handle • **Tamper:** Selected tamper pin. This parameter can be any combination of RTC_TAMPER_1, RTC_TAMPER_2 and RTC_TAMPER_3.
Return values	• **HAL:** status

HAL_RTCEx_TamperTimeStampIRQHandler

Function name	**void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)**
Function description	Handle TimeStamp interrupt request.
Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTCEx_Tamper1EventCallback

Function name	**void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Tamper 1 callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTCEx_Tamper2EventCallback

Function name	**void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Tamper 2 callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTCEx_Tamper3EventCallback

Function name	**void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Tamper 3 callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTCEx_TimeStampEventCallback

Function name	**void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)**
Function description	TimeStamp callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTCEx_PollForTimeStampEvent

Function name	**HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle TimeStamp polling request.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_RTCEx_PollForTamper1Event

Function name	**HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle Tamper 1 Polling.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_RTCEx_PollForTamper2Event

Function name	**HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle Tamper 2 Polling.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_RTCEx_PollForTamper3Event

Function name	**HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle Tamper 3 Polling.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_RTCEx_SetWakeUpTimer

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)**
Function description	Set wake up timer.
Parameters	• **hrtc:** RTC handle • **WakeUpCounter:** Wake up counter • **WakeUpClock:** Wake up clock
Return values	• **HAL:** status

HAL_RTCEx_SetWakeUpTimer_IT

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)**
Function description	Set wake up timer with interrupt.
Parameters	• **hrtc:** RTC handle • **WakeUpCounter:** Wake up counter • **WakeUpClock:** Wake up clock
Return values	• **HAL:** status

HAL_RTCEx_DeactivateWakeUpTimer

Function name	**uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)**
Function description	Deactivate wake up timer counter.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTCEx_GetWakeUpTimer

Function name	**uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)**
Function description	Get wake up timer counter.
Parameters	• **hrtc:** RTC handle
Return values	• **Counter:** value

HAL_RTCEx_WakeUpTimerIRQHandler

Function name	**void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)**
Function description	Handle Wake Up Timer interrupt request.
Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTCEx_WakeUpTimerEventCallback

Function name	**void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Wake Up Timer callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTCEx_PollForWakeUpTimerEvent

Function name	**HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle Wake Up Timer Polling.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_RTCEx_BKUPWrite

Function name	**void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)**
Function description	Write a data in a specified RTC Backup data register.
Parameters	• **hrtc:** RTC handle • **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register. • **Data:** Data to be written in the specified RTC Backup data register.
Return values	• **None:**

HAL_RTCEx_BKUPRead

Function name	**uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)**
Function description	Read data from the specified RTC Backup data Register.
Parameters	• **hrtc:** RTC handle • **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.
Return values	• **Read:** value

HAL_RTCEx_SetSmoothCalib

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)**

Function description	Set the Smooth calibration parameters.
Parameters	• **hrtc:** RTC handle • **SmoothCalibPeriod:** Select the Smooth Calibration Period. This parameter can be can be one of the following values: – RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s. – RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s. – RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s. • **SmoothCalibPlusPulses:** Select to Set or reset the CALP bit. This parameter can be one of the following values: – RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulse every 2*11 pulses. – RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added. • **SmoothCalibMinusPulsesValue:** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.
Return values	• **HAL:** status
Notes	• To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

HAL_RTCEx_SetSynchroShift

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)**
Function description	Configure the Synchronization Shift Control Settings.
Parameters	• **hrtc:** RTC handle • **ShiftAdd1S:** Select to add or not 1 second to the time calendar. This parameter can be one of the following values: – RTC_SHIFTADD1S_SET: Add one second to the clock calendar. – RTC_SHIFTADD1S_RESET: No effect. • **ShiftSubFS:** Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.
Return values	• **HAL:** status
Notes	• When REFCKON is set, firmware must not write to Shift control register.

HAL_RTCEx_SetCalibrationOutPut

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)**
Function description	Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or

512Hz).

Parameters	• **hrtc:** RTC handle • **CalibOutput:** : Select the Calibration output Selection . This parameter can be one of the following values: – RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz. – RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.
Return values	• **HAL:** status

HAL_RTCEx_DeactivateCalibrationOutPut

Function name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)**
Function description	Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTCEx_SetRefClock

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)**
Function description	Enable the RTC reference clock detection.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTCEx_DeactivateRefClock

Function name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)**
Function description	Disable the RTC reference clock detection.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status

HAL_RTCEx_EnableBypassShadow

Function name	**HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)**
Function description	Enable the Bypass Shadow feature.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status
Notes	• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_DisableBypassShadow

Function name	**HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)**
Function description	Disable the Bypass Shadow feature.
Parameters	• **hrtc:** RTC handle
Return values	• **HAL:** status
Notes	• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_AlarmBEventCallback

Function name	**void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)**
Function description	Alarm B callback.
Parameters	• **hrtc:** RTC handle
Return values	• **None:**

HAL_RTCEx_PollForAlarmBEvent

Function name	**HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function description	Handle Alarm B Polling request.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• **HAL:** status

55.3 RTCEx Firmware driver defines

55.3.1 RTCEx

RTC Add 1 Second Parameter Definitions

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

RTC Backup Registers Definitions

RTC_BKP_DR0

RTC_BKP_DR1

RTC_BKP_DR2

RTC_BKP_DR3

RTC_BKP_DR4

RTC_BKP_DR5

RTC_BKP_DR6

RTC_BKP_DR7

RTC_BKP_DR8

RTC_BKP_DR9

RTC_BKP_DR10

RTC_BKP_DR11

RTC_BKP_DR12

RTC_BKP_DR13

RTC_BKP_DR14

RTC_BKP_DR15

RTC_BKP_DR16

RTC_BKP_DR17

RTC_BKP_DR18

RTC_BKP_DR19

RTC_BKP_DR20

RTC_BKP_DR21

RTC_BKP_DR22

RTC_BKP_DR23

RTC_BKP_DR24

RTC_BKP_DR25

RTC_BKP_DR26

RTC_BKP_DR27

RTC_BKP_DR28

RTC_BKP_DR29

RTC_BKP_DR30

RTC_BKP_DR31

RTC Calib Output Selection Definitions

RTC_CALIBOUTPUT_512HZ

RTC_CALIBOUTPUT_1HZ

RTCEx Exported Macros

__HAL_RTC_WAKEUPTIMER_ENABLE	**Description:** • Enable the RTC WakeUp Timer peripheral. **Parameters:** • __HANDLE__: specifies the RTC handle. **Return value:** • None
__HAL_RTC_WAKEUPTIMER_DISABLE	**Description:**

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_ENABLE_IT

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
  - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE_IT

Description:

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be disabled. This parameter can be:
  - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT

Description:

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt

sources to check. This parameter can be:
   - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Wake Up timer interrupt is enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
   - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_FLAG

Description:

- Get the selected RTC WakeUpTimer's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
   - RTC_FLAG_WUTF
   - RTC_FLAG_WUTWF

Return value:

- None

__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG

Description:

- Clear the RTC Wake Up timer's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
   - RTC_FLAG_WUTF

Return value:

- None

__HAL_RTC_TAMPER1_ENABLE

Description:

- Enable the RTC Tamper1 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER1_DISABLE

Description:

- Disable the RTC Tamper1 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER2_ENABLE

Description:

- Enable the RTC Tamper2 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER2_DISABLE

Description:

- Disable the RTC Tamper2 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER3_ENABLE

Description:

- Enable the RTC Tamper3 input detection.

Parameters:

- __HANDLE__: specifies the

RTC handle.

Return value:

- None

__HAL_RTC_TAMPER3_DISABLE

Description:

- Disable the RTC Tamper3 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER_ENABLE_IT

Description:

- Enable the RTC Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP: All tampers interrupts
  - RTC_IT_TAMP1: Tamper1 interrupt
  - RTC_IT_TAMP2: Tamper2 interrupt
  - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_DISABLE_IT

Description:

- Disable the RTC Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP: All tampers interrupts

  - RTC_IT_TAMP1: Tamper1 interrupt
  - RTC_IT_TAMP2: Tamper2 interrupt
  - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_IT

Description:

- Check whether the specified RTC Tamper interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt to check. This parameter can be:
  - RTC_IT_TAMP1: Tamper1 interrupt
  - RTC_IT_TAMP2: Tamper2 interrupt
  - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Tamper interrupt is enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt source to check. This parameter can be:
  - RTC_IT_TAMP: All tampers interrupts
  - RTC_IT_TAMP1: Tamper1 interrupt
  - RTC_IT_TAMP2: Tamper2 interrupt
  - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_FLAG

Description:

- Get the selected RTC Tamper's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag is pending or not. This parameter can be:
  - RTC_FLAG_TAMP1F: Tamper1 flag
  - RTC_FLAG_TAMP2F: Tamper2 flag
  - RTC_FLAG_TAMP3F: Tamper3 flag

Return value:

- None

__HAL_RTC_TAMPER_CLEAR_FLAG

Description:

- Clear the RTC Tamper's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag sources to clear. This parameter can be:
  - RTC_FLAG_TAMP1F: Tamper1 flag
  - RTC_FLAG_TAMP2F: Tamper2 flag
  - RTC_FLAG_TAMP3F: Tamper3 flag

Return value:

- None

__HAL_RTC_TIMESTAMP_ENABLE

Description:

- Enable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE

Description:

- Disable the RTC TimeStamp

peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_ENABLE_IT

Description:

- Enable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be:
  - RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE_IT

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be:
  - RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT

Description:

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to check. This parameter

can be:
  – RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT_SOURCE

Description:

- Check whether the specified RTC Time Stamp interrupt is enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
  – RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC TimeStamp's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
  – RTC_FLAG_TSF
  – RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Time Stamp's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  – RTC_FLAG_TSF
  – RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_ENABLE

Description:

- Enable the RTC internal TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_DISABLE

Description:

- Disable the RTC internal TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC Internal Time Stamp's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag is pending or not. This parameter can be:
  - RTC_FLAG_ITSF

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Internal Time Stamp's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag source to clear. This parameter can be:
  - RTC_FLAG_ITSF

Return value:

- None

__HAL_RTC_CALIBRATION_OUTPUT_ENABLE

Description:

- Enable the RTC calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CALIBRATION_OUTPUT_DISABLE

Description:

- Disable the calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CLOCKREF_DETECTION_ENABLE

Description:

- Enable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CLOCKREF_DETECTION_DISABLE

Description:

- Disable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_SHIFT_GET_FLAG

Description:

- Get the selected RTC shift operation's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.

- __FLAG__: specifies the RTC shift operation Flag is pending or not. This parameter can be:
  - RTC_FLAG_SHPF

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_EDGE

Description:

- Enable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_EDGE

Description:

- Disable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_GET_FLAG

Description:

- Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

__HAL_RTC_WAKEUPTIMER_EXTI_CLEAR_FLAG

Description:

- Clear the RTC WakeUp Timer associated Exti line flag.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_EDGE

Description:

- Enable rising edge trigger on the RTC Tamper and

Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_EDGE

Description:

- Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GET_FLAG

Description:

- Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_CLEAR_FLAG

Description:

- Clear the RTC Tamper and Timestamp associated Exti line flag.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

Private macros to check input parameters

IS_RTC_OUTPUT

IS_RTC_BKP

IS_TIMESTAMP_EDGE

IS_RTC_TAMPER

IS_RTC_TAMPER_INTERRUPT

IS_RTC_TIMESTAMP_PIN

IS_RTC_TAMPER_TRIGGER

IS_RTC_TAMPER_ERASE_MODE

IS_RTC_TAMPER_MASKFLAG_STATE

IS_RTC_TAMPER_FILTER

IS_RTC_TAMPER_SAMPLING_FREQ

IS_RTC_TAMPER_PRECHARGE_DURATION

IS_RTC_TAMPER_TIMESTAMPONTAMPER_DETECTION

IS_RTC_TAMPER_PULLUP_STATE

IS_RTC_WAKEUP_CLOCK

IS_RTC_WAKEUP_COUNTER

IS_RTC_SMOOTH_CALIB_PERIOD

IS_RTC_SMOOTH_CALIB_PLUS

IS_RTC_SMOOTH_CALIB_MINUS

IS_RTC_SHIFT_ADD1S

IS_RTC_SHIFT_SUBFS

IS_RTC_CALIB_OUTPUT

RTC Output Selection Definitions

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTC_OUTPUT_ALARMB

RTC_OUTPUT_WAKEUP

RTC Smooth Calib Period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK seconds
RTC_SMOOTHCALIB_PERIOD_16SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK seconds
RTC_SMOOTHCALIB_PERIOD_8SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK seconds

RTC Smooth Calib Plus Pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET	The number of RTCCLK pulses added during a X -second window = Y -

	CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8
RTC_SMOOTHCALIB_PLUSPULSES_RESET	The number of RTCCLK pulses subbstited during a 32-second window = CALM[8:0]

RTC Tamper EraseBackUp Definitions

RTC_TAMPER_ERASE_BACKUP_ENABLE

RTC_TAMPER_ERASE_BACKUP_DISABLE

RTC Tamper Filter Definitions

RTC_TAMPERFILTER_DISABLE	Tamper filter is disabled
RTC_TAMPERFILTER_2SAMPLE	Tamper is activated after 2 consecutive samples at the active level
RTC_TAMPERFILTER_4SAMPLE	Tamper is activated after 4 consecutive samples at the active level
RTC_TAMPERFILTER_8SAMPLE	Tamper is activated after 8 consecutive samples at the active level.

RTC Tamper Interrupts Definitions

RTC_TAMPER1_INTERRUPT

RTC_TAMPER2_INTERRUPT

RTC_TAMPER3_INTERRUPT

RTC_ALL_TAMPER_INTERRUPT

RTC Tamper Mask Flag Definitions

RTC_TAMPERMASK_FLAG_DISABLE

RTC_TAMPERMASK_FLAG_ENABLE

RTC Tamper Pins Definitions

RTC_TAMPER_1

RTC_TAMPER_2

RTC_TAMPER_3

RTC Tamper Pin Precharge Duration Definitions

RTC_TAMPERPRECHARGEDURATION_1RTCCLK	Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
RTC_TAMPERPRECHARGEDURATION_2RTCCLK	Tamper pins are pre-charged before sampling during 2 RTCCLK cycles
RTC_TAMPERPRECHARGEDURATION_4RTCCLK	Tamper pins are pre-charged before sampling during 4 RTCCLK cycles
RTC_TAMPERPRECHARGEDURATION_8RTCCLK	Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

RTC Tamper Pull Up Definitions

RTC_TAMPER_PULLUP_ENABLE	TimeStamp on Tamper Detection event saved
RTC_TAMPER_PULLUP_DISABLE	TimeStamp on Tamper Detection event is not saved

RTC Tamper Sampling Frequencies Definitions

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768	Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384	Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192	Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024	Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

RTC Tamper TimeStamp On Tamper Detection Definitions

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE	TimeStamp on Tamper Detection event saved
RTC_TIMESTAMPONTAMPERDETECTION_DISABLE	TimeStamp on Tamper Detection event is not saved

RTC Tamper Triggers Definitions

RTC_TAMPERTRIGGER_RISINGEDGE

RTC_TAMPERTRIGGER_FALLINGEDGE

RTC_TAMPERTRIGGER_LOWLEVEL

RTC_TAMPERTRIGGER_HIGHLEVEL

RTC TimeStamp Edges Definitions

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

RTC TimeStamp Pins Selection

RTC_TIMESTAMPPIN_DEFAULT

RTC Wakeup Timer Definitions

RTC_WAKEUPCLOCK_RTCCLK_DIV16

RTC_WAKEUPCLOCK_RTCCLK_DIV8

RTC_WAKEUPCLOCK_RTCCLK_DIV4

RTC_WAKEUPCLOCK_RTCCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

56 HAL SAI Generic Driver

56.1 SAI Firmware driver registers structures

56.1.1 SAI_PdmInitTypeDef

Data Fields

- ***FunctionalState Activation***
- ***uint32_t MicPairsNbr***
- ***uint32_t ClockEnable***

Field Documentation

- ***FunctionalState SAI_PdmInitTypeDef::Activation***
  Enable/disable PDM interface
- ***uint32_t SAI_PdmInitTypeDef::MicPairsNbr***
  Specifies the number of microphone pairs used. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAI_PdmInitTypeDef::ClockEnable***
  Specifies which clock must be enabled. This parameter can be a values combination of ***SAI_PDM_ClockEnable***

56.1.2 SAI_InitTypeDef

Data Fields

- ***uint32_t AudioMode***
- ***uint32_t Synchro***
- ***uint32_t SynchroExt***
- ***uint32_t OutputDrive***
- ***uint32_t NoDivider***
- ***uint32_t FIFOThreshold***
- ***uint32_t AudioFrequency***
- ***uint32_t Mckdiv***
- ***uint32_t MckOverSampling***
- ***uint32_t MonoStereoMode***
- ***uint32_t CompandingMode***
- ***uint32_t TriState***
- ***SAI_PdmInitTypeDef PdmInit***
- ***uint32_t Protocol***
- ***uint32_t DataSize***
- ***uint32_t FirstBit***
- ***uint32_t ClockStrobing***

Field Documentation

- ***uint32_t SAI_InitTypeDef::AudioMode***
  Specifies the SAI Block audio Mode. This parameter can be a value of ***SAI_Block_Mode***
- ***uint32_t SAI_InitTypeDef::Synchro***
  Specifies SAI Block synchronization This parameter can be a value of ***SAI_Block_Synchronization***

- ***uint32_t SAI_InitTypeDef::SynchroExt***
  Specifies SAI external output synchronization, this setup is common for BlockA and BlockB This parameter can be a value of ***SAI_Block_SyncExt***
  Note:: If both audio blocks of same SAI are used, this parameter has to be set to the same value for each audio block
- ***uint32_t SAI_InitTypeDef::OutputDrive***
  Specifies when SAI Block outputs are driven. This parameter can be a value of ***SAI_Block_Output_Drive***
  Note:this value has to be set before enabling the audio block but after the audio block configuration.
- ***uint32_t SAI_InitTypeDef::NoDivider***
  Specifies whether master clock will be divided or not. This parameter can be a value of ***SAI_Block_NoDivider***
  Note:: For STM32L4Rx/STM32L4Sx devices: If bit NOMCK in the SAI_xCR1 register is cleared, the frame length should be aligned to a number equal to a power of 2, from 8 to 256. If bit NOMCK in the SAI_xCR1 register is set, the frame length can take any of the values without constraint. There is no MCLK_x clock which can be output. For other devices: If bit NODIV in the SAI_xCR1 register is cleared, the frame length should be aligned to a number equal to a power of 2, from 8 to 256. If bit NODIV in the SAI_xCR1 register is set, the frame length can take any of the values without constraint since the input clock of the audio block should be equal to the bit clock. There is no MCLK_x clock which can be output.
- ***uint32_t SAI_InitTypeDef::FIFOThreshold***
  Specifies SAI Block FIFO threshold. This parameter can be a value of ***SAI_Block_Fifo_Threshold***
- ***uint32_t SAI_InitTypeDef::AudioFrequency***
  Specifies the audio frequency sampling. This parameter can be a value of ***SAI_Audio_Frequency***
- ***uint32_t SAI_InitTypeDef::Mckdiv***
  Specifies the master clock divider, the parameter will be used if for AudioFrequency the user choice This parameter must be a number between Min_Data = 0 and Max_Data = 63 on STM32L4Rx/STM32L4Sx devices. This parameter must be a number between Min_Data = 0 and Max_Data = 15 on other devices.
- ***uint32_t SAI_InitTypeDef::MckOverSampling***
  Specifies the master clock oversampling. This parameter can be a value of ***SAI_Block_Mck_OverSampling***
- ***uint32_t SAI_InitTypeDef::MonoStereoMode***
  Specifies if the mono or stereo mode is selected. This parameter can be a value of ***SAI_Mono_Stereo_Mode***
- ***uint32_t SAI_InitTypeDef::CompandingMode***
  Specifies the companding mode type. This parameter can be a value of ***SAI_Block_Companding_Mode***
- ***uint32_t SAI_InitTypeDef::TriState***
  Specifies the companding mode type. This parameter can be a value of ***SAI_TRIState_Management***
- ***SAI_PdmInitTypeDef SAI_InitTypeDef::PdmInit***
  Specifies the PDM configuration.
- ***uint32_t SAI_InitTypeDef::Protocol***
  Specifies the SAI Block protocol. This parameter can be a value of ***SAI_Block_Protocol***
- ***uint32_t SAI_InitTypeDef::DataSize***
  Specifies the SAI Block data size. This parameter can be a value of ***SAI_Block_Data_Size***

- ***uint32_t SAI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SAI_Block_MSB_LSB_transmission***
- ***uint32_t SAI_InitTypeDef::ClockStrobing***
  Specifies the SAI Block clock strobing edge sensitivity. This parameter can be a value of ***SAI_Block_Clock_Strobing***

56.1.3 SAI_FrameInitTypeDef

Data Fields

- ***uint32_t FrameLength***
- ***uint32_t ActiveFrameLength***
- ***uint32_t FSDefinition***
- ***uint32_t FSPolarity***
- ***uint32_t FSOffset***

Field Documentation

- ***uint32_t SAI_FrameInitTypeDef::FrameLength***
  Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min_Data = 8 and Max_Data = 256.
  Note:: If master clock MCLK_x pin is declared as an output, the frame length should be aligned to a number equal to power of 2 in order to keep in an audio frame, an integer number of MCLK pulses by bit Clock.
- ***uint32_t SAI_FrameInitTypeDef::ActiveFrameLength***
  Specifies the Frame synchronization active level length. This Parameter specifies the length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t SAI_FrameInitTypeDef::FSDefinition***
  Specifies the Frame synchronization definition. This parameter can be a value of ***SAI_Block_FS_Definition***
- ***uint32_t SAI_FrameInitTypeDef::FSPolarity***
  Specifies the Frame synchronization Polarity. This parameter can be a value of ***SAI_Block_FS_Polarity***
- ***uint32_t SAI_FrameInitTypeDef::FSOffset***
  Specifies the Frame synchronization Offset. This parameter can be a value of ***SAI_Block_FS_Offset***

56.1.4 SAI_SlotInitTypeDef

Data Fields

- ***uint32_t FirstBitOffset***
- ***uint32_t SlotSize***
- ***uint32_t SlotNumber***
- ***uint32_t SlotActive***

Field Documentation

- ***uint32_t SAI_SlotInitTypeDef::FirstBitOffset***
  Specifies the position of first data transfer bit in the slot. This parameter must be a number between Min_Data = 0 and Max_Data = 24
- ***uint32_t SAI_SlotInitTypeDef::SlotSize***
  Specifies the Slot Size. This parameter can be a value of ***SAI_Block_Slot_Size***
- ***uint32_t SAI_SlotInitTypeDef::SlotNumber***
  Specifies the number of slot in the audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 16

- ***uint32_t SAI_SlotInitTypeDef::SlotActive***
  Specifies the slots in audio frame that will be activated. This parameter can be a value of ***SAI_Block_Slot_Active***

56.1.5 __SAI_HandleTypeDef

Data Fields

- ***SAI_Block_TypeDef * Instance***
- ***SAI_InitTypeDef Init***
- ***SAI_FrameInitTypeDef FrameInit***
- ***SAI_SlotInitTypeDef SlotInit***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***uint16_t XferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***SAIcallback mutecallback***
- ***void(* InterruptServiceRoutine***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SAI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***SAI_Block_TypeDef* __SAI_HandleTypeDef::Instance***
  SAI Blockx registers base address
- ***SAI_InitTypeDef __SAI_HandleTypeDef::Init***
  SAI communication parameters
- ***SAI_FrameInitTypeDef __SAI_HandleTypeDef::FrameInit***
  SAI Frame configuration parameters
- ***SAI_SlotInitTypeDef __SAI_HandleTypeDef::SlotInit***
  SAI Slot configuration parameters
- ***uint8_t* __SAI_HandleTypeDef::pBuffPtr***
  Pointer to SAI transfer Buffer
- ***uint16_t __SAI_HandleTypeDef::XferSize***
  SAI transfer size
- ***uint16_t __SAI_HandleTypeDef::XferCount***
  SAI transfer counter
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmatx***
  SAI Tx DMA handle parameters
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmarx***
  SAI Rx DMA handle parameters
- ***SAIcallback __SAI_HandleTypeDef::mutecallback***
  SAI mute callback
- ***void(* __SAI_HandleTypeDef::InterruptServiceRoutine)(struct __SAI_HandleTypeDef *hsai)***
- ***HAL_LockTypeDef __SAI_HandleTypeDef::Lock***
  SAI locking object
- ***__IO HAL_SAI_StateTypeDef __SAI_HandleTypeDef::State***
  SAI communication state
- ***__IO uint32_t __SAI_HandleTypeDef::ErrorCode***
  SAI Error code

56.2 SAI Firmware driver API description

56.2.1 How to use this driver

The SAI HAL driver can be used as follows:

1. Declare a SAI_HandleTypeDef handle structure (eg. SAI_HandleTypeDef hsai).
2. Initialize the SAI low level resources by implementing the HAL_SAI_MspInit() API:
   a. Enable the SAI interface clock.
   b. SAI pins configuration:
      – Enable the clock for the SAI GPIOs.
      – Configure these SAI pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_SAI_Transmit_IT() and HAL_SAI_Receive_IT() APIs):
      – Configure the SAI interrupt priority.
      – Enable the NVIC SAI IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_SAI_Transmit_DMA() and HAL_SAI_Receive_DMA() APIs):
      – Declare a DMA handle structure for the Tx/Rx stream.
      – Enable the DMAx interface clock.
      – Configure the declared DMA handle structure with the required Tx/Rx parameters.
      – Configure the DMA Tx/Rx Stream.
      – Associate the initialized DMA handle to the SAI DMA Tx/Rx handle.
      – Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. The initialization can be done by two ways
   a. Expert mode: Initialize the structures Init, FrameInit and SlotInit and call HAL_SAI_Init().
   b. Simplified mode: Initialize the high part of Init Structure and call HAL_SAI_InitProtocol().

The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros __HAL_SAI_ENABLE_IT() and __HAL_SAI_DISABLE_IT() inside the transmit and receive process.

Make sure that either:

- PLLSAI1CLK output is configured or
- PLLSAI2CLK output is configured or
- PLLSAI3CLK output is configured or
- External clock source is configured after setting correctly the define constant EXTERNAL_SAI1_CLOCK_VALUE or EXTERNAL_SAI2_CLOCK_VALUE in the stm32l4xx_hal_conf.h file.

In master Tx mode: enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO, However FS signal generation is conditioned by the presence of data in the FIFO.

In master Rx mode: enabling the audio block immediately generates the bit clock and FS signal for the external slaves.

It is mandatory to respect the following conditions in order to avoid bad SAI behavior:

- First bit Offset <= (SLOT size - Data size)
- Data size <= SLOT size
- Number of SLOT x SLOT size = Frame length
- The number of slots should be even when SAI_FS_CHANNEL_IDENTIFICATION is selected.

For STM32L4Rx/STM32L4Sx devices, PDM interface can be activated through HAL_SAI_Init function. Please note that PDM interface is only available for SAI1 sub-block A. PDM microphone delays can be tuned with HAL_SAIEx_ConfigPdmMicDelay function.

Three operation modes are available within this driver:

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SAI_Transmit()
- Receive an amount of data in blocking mode using HAL_SAI_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SAI_Transmit_IT()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SAI_Receive_IT()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SAI_Transmit_DMA()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SAI_Receive_DMA()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()
- Pause the DMA Transfer using HAL_SAI_DMAPause()
- Resume the DMA Transfer using HAL_SAI_DMAResume()
- Stop the DMA Transfer using HAL_SAI_DMAStop()

SAI HAL driver additional function list

Below the list the others API available SAI HAL driver:

- HAL_SAI_EnableTxMuteMode(): Enable the mute in tx mode
- HAL_SAI_DisableTxMuteMode(): Disable the mute in tx mode
- HAL_SAI_EnableRxMuteMode(): Enable the mute in Rx mode
- HAL_SAI_DisableRxMuteMode(): Disable the mute in Rx mode
- HAL_SAI_FlushRxFifo(): Flush the rx fifo.
- HAL_SAI_Abort(): Abort the current transfer

SAI HAL driver macros list

Below the list of most used macros in SAI HAL driver:

- __HAL_SAI_ENABLE(): Enable the SAI peripheral
- __HAL_SAI_DISABLE(): Disable the SAI peripheral
- __HAL_SAI_ENABLE_IT(): Enable the specified SAI interrupts
- __HAL_SAI_DISABLE_IT(): Disable the specified SAI interrupts
- __HAL_SAI_GET_IT_SOURCE(): Check if the specified SAI interrupt source is enabled or disabled
- __HAL_SAI_GET_FLAG(): Check whether the specified SAI flag is set or not

56.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SAIx peripheral:

- User must implement HAL_SAI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SAI_Init() to configure the selected device with the selected configuration:
  - Mode (Master/slave TX/RX)
  - Protocol
  - Data Size
  - MCLK Output
  - Audio frequency
  - FIFO Threshold
  - Frame Config
  - Slot Config
  - PDM Config (only for STM32L4Rx/STM32L4Sx devices)
- Call the function HAL_SAI_DeInit() to restore the default configuration of the selected SAI peripheral.

This section contains the following APIs:

- ***HAL_SAI_InitProtocol()***
- ***HAL_SAI_Init()***
- ***HAL_SAI_DeInit()***
- ***HAL_SAI_MspInit()***
- ***HAL_SAI_MspDeInit()***

56.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SAI data transfers.

- There are two modes of transfer:

  - Blocking mode: The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SAI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
- Blocking mode functions are:
  - HAL_SAI_Transmit()
  - HAL_SAI_Receive()
- Non Blocking mode functions with Interrupt are:
  - HAL_SAI_Transmit_IT()
  - HAL_SAI_Receive_IT()
- Non Blocking mode functions with DMA are:
  - HAL_SAI_Transmit_DMA()
  - HAL_SAI_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL_SAI_TxCpltCallback()
  - HAL_SAI_RxCpltCallback()
  - HAL_SAI_ErrorCallback()

This section contains the following APIs:

- ***HAL_SAI_Transmit()***
- ***HAL_SAI_Receive()***
- ***HAL_SAI_Transmit_IT()***
- ***HAL_SAI_Receive_IT()***
- ***HAL_SAI_DMAPause()***
- ***HAL_SAI_DMAResume()***
- ***HAL_SAI_DMAStop()***
- ***HAL_SAI_Abort()***
- ***HAL_SAI_Transmit_DMA()***
- ***HAL_SAI_Receive_DMA()***
- ***HAL_SAI_EnableTxMuteMode()***
- ***HAL_SAI_DisableTxMuteMode()***
- ***HAL_SAI_EnableRxMuteMode()***
- ***HAL_SAI_DisableRxMuteMode()***
- ***HAL_SAI_IRQHandler()***
- ***HAL_SAI_TxCpltCallback()***
- ***HAL_SAI_TxHalfCpltCallback()***
- ***HAL_SAI_RxCpltCallback()***
- ***HAL_SAI_RxHalfCpltCallback()***
- ***HAL_SAI_ErrorCallback()***

56.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_SAI_GetState()***
- ***HAL_SAI_GetError()***

56.2.5 Detailed description of functions

HAL_SAI_InitProtocol

Function name	**HAL_StatusTypeDef HAL_SAI_InitProtocol (SAI_HandleTypeDef * hsai, uint32_t protocol, uint32_t datasize, uint32_t nbslot)**
Function description	Initialize the structure FrameInit, SlotInit and the low part of Init according to the specified parameters and call the function HAL_SAI_Init to initialize the SAI block.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • **protocol:** one of the supported protocol SAI Supported protocol • **datasize:** one of the supported datasize SAI protocol data size the configuration information for SAI module. • **nbslot:** Number of slot.
Return values	• **HAL:** status

HAL_SAI_Init

Function name	**HAL_StatusTypeDef HAL_SAI_Init (SAI_HandleTypeDef * hsai)**
Function description	Initialize the SAI according to the specified parameters.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL:** status

HAL_SAI_DeInit

Function name	**HAL_StatusTypeDef HAL_SAI_DeInit (SAI_HandleTypeDef * hsai)**
Function description	DeInitialize the SAI peripheral.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL:** status

HAL_SAI_MspInit

Function name	**void HAL_SAI_MspInit (SAI_HandleTypeDef * hsai)**
Function description	Initialize the SAI MSP.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **None:**

HAL_SAI_MspDeInit

Function name	**void HAL_SAI_MspDeInit (SAI_HandleTypeDef * hsai)**

Function description	DeInitialize the SAI MSP.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **None:**

HAL_SAI_Transmit

Function name	**HAL_StatusTypeDef HAL_SAI_Transmit (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Transmit an amount of data in blocking mode.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_SAI_Receive

Function name	**HAL_StatusTypeDef HAL_SAI_Receive (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be received • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_SAI_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_SAI_Transmit_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)**
Function description	Transmit an amount of data in non-blocking mode with Interrupt.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_SAI_Receive_IT

Function name	**HAL_StatusTypeDef HAL_SAI_Receive_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in non-blocking mode with Interrupt.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values	• **HAL:** status

HAL_SAI_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_SAI_Transmit_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)**
Function description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• **HAL:** status

HAL_SAI_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_SAI_Receive_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in non-blocking mode with DMA.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be received
Return values	• **HAL:** status

HAL_SAI_DMAPause

Function name	**HAL_StatusTypeDef HAL_SAI_DMAPause (SAI_HandleTypeDef * hsai)**
Function description	Pause the audio stream playing from the Media.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL:** status

HAL_SAI_DMAResume

Function name	**HAL_StatusTypeDef HAL_SAI_DMAResume (SAI_HandleTypeDef * hsai)**
Function description	Resume the audio stream playing from the Media.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL:** status

HAL_SAI_DMAStop

Function name	**HAL_StatusTypeDef HAL_SAI_DMAStop (SAI_HandleTypeDef * hsai)**
Function description	Stop the audio stream playing from the Media.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL:** status

HAL_SAI_Abort

Function name	**HAL_StatusTypeDef HAL_SAI_Abort (SAI_HandleTypeDef * hsai)**
Function description	Abort the current transfer and disable the SAI.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL:** status

HAL_SAI_EnableTxMuteMode

Function name	**HAL_StatusTypeDef HAL_SAI_EnableTxMuteMode (SAI_HandleTypeDef * hsai, uint16_t val)**
Function description	Enable the Tx mute mode.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • **val:** value sent during the mute SAI Block Mute Value
Return values	• **HAL:** status

HAL_SAI_DisableTxMuteMode

Function name	**HAL_StatusTypeDef HAL_SAI_DisableTxMuteMode (SAI_HandleTypeDef * hsai)**
Function description	Disable the Tx mute mode.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL:** status

HAL_SAI_EnableRxMuteMode

Function name	**HAL_StatusTypeDef HAL_SAI_EnableRxMuteMode (SAI_HandleTypeDef * hsai, SAIcallback callback, uint16_t counter)**
Function description	Enable the Rx mute detection.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • **callback:** function called when the mute is detected. • **counter:** number a data before mute detection max 63.

Return values	• **HAL:** status

HAL_SAI_DisableRxMuteMode

Function name	**HAL_StatusTypeDef HAL_SAI_DisableRxMuteMode (SAI_HandleTypeDef * hsai)**
Function description	Disable the Rx mute detection.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL:** status

HAL_SAI_IRQHandler

Function name	**void HAL_SAI_IRQHandler (SAI_HandleTypeDef * hsai)**
Function description	Handle SAI interrupt request.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **None:**

HAL_SAI_TxHalfCpltCallback

Function name	**void HAL_SAI_TxHalfCpltCallback (SAI_HandleTypeDef * hsai)**
Function description	Tx Transfer Half completed callback.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **None:**

HAL_SAI_TxCpltCallback

Function name	**void HAL_SAI_TxCpltCallback (SAI_HandleTypeDef * hsai)**
Function description	Tx Transfer completed callback.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **None:**

HAL_SAI_RxHalfCpltCallback

Function name	**void HAL_SAI_RxHalfCpltCallback (SAI_HandleTypeDef * hsai)**
Function description	Rx Transfer half completed callback.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **None:**

HAL_SAI_RxCpltCallback

Function name	**void HAL_SAI_RxCpltCallback (SAI_HandleTypeDef * hsai)**
Function description	Rx Transfer completed callback.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **None:**

HAL_SAI_ErrorCallback

Function name	**void HAL_SAI_ErrorCallback (SAI_HandleTypeDef * hsai)**
Function description	SAI error callback.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **None:**

HAL_SAI_GetState

Function name	**HAL_SAI_StateTypeDef HAL_SAI_GetState (SAI_HandleTypeDef * hsai)**
Function description	Return the SAI handle state.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL:** state

HAL_SAI_GetError

Function name	**uint32_t HAL_SAI_GetError (SAI_HandleTypeDef * hsai)**
Function description	Return the SAI error code.
Parameters	• **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for the specified SAI Block.
Return values	• **SAI:** Error Code

56.3 SAI Firmware driver defines

56.3.1 SAI

SAI Audio Frequency

SAI_AUDIO_FREQUENCY_192K

SAI_AUDIO_FREQUENCY_96K

SAI_AUDIO_FREQUENCY_48K

SAI_AUDIO_FREQUENCY_44K

SAI_AUDIO_FREQUENCY_32K

SAI_AUDIO_FREQUENCY_22K

SAI_AUDIO_FREQUENCY_16K

SAI_AUDIO_FREQUENCY_11K

SAI_AUDIO_FREQUENCY_8K

SAI_AUDIO_FREQUENCY_MCKDIV

SAI Block Clock Strobing

SAI_CLOCKSTROBING_FALLINGEDGE

SAI_CLOCKSTROBING_RISINGEDGE

SAI Block Companding Mode

SAI_NOCOMPANDING

SAI_ULAW_1CPL_COMPANDING

SAI_ALAW_1CPL_COMPANDING

SAI_ULAW_2CPL_COMPANDING

SAI_ALAW_2CPL_COMPANDING

SAI Block Data Size

SAI_DATASIZE_8

SAI_DATASIZE_10

SAI_DATASIZE_16

SAI_DATASIZE_20

SAI_DATASIZE_24

SAI_DATASIZE_32

SAI Block Fifo Status Level

SAI_FIFOSTATUS_EMPTY

SAI_FIFOSTATUS_LESS1QUARTERFULL

SAI_FIFOSTATUS_1QUARTERFULL

SAI_FIFOSTATUS_HALFFULL

SAI_FIFOSTATUS_3QUARTERFULL

SAI_FIFOSTATUS_FULL

SAI Block Fifo Threshold

SAI_FIFOTHRESHOLD_EMPTY

SAI_FIFOTHRESHOLD_1QF

SAI_FIFOTHRESHOLD_HF

SAI_FIFOTHRESHOLD_3QF

SAI_FIFOTHRESHOLD_FULL

SAI Block Flags Definition

SAI_FLAG_OVRUDR

SAI_FLAG_MUTEDET

SAI_FLAG_WCKCFG

SAI_FLAG_FREQ

SAI_FLAG_CNRDY

SAI_FLAG_AFSDET

SAI_FLAG_LFSDET

SAI Block FS Definition

SAI_FS_STARTFRAME

SAI_FS_CHANNEL_IDENTIFICATION

SAI Block FS Offset

SAI_FS_FIRSTBIT

SAI_FS_BEFOREFIRSTBIT

SAI Block FS Polarity

SAI_FS_ACTIVE_LOW

SAI_FS_ACTIVE_HIGH

SAI Block Interrupts Definition

SAI_IT_OVRUDR

SAI_IT_MUTEDET

SAI_IT_WCKCFG

SAI_IT_FREQ

SAI_IT_CNRDY

SAI_IT_AFSDET

SAI_IT_LFSDET

SAI Block Master Clock OverSampling

SAI_MCK_OVERSAMPLING_DISABLE

SAI_MCK_OVERSAMPLING_ENABLE

SAI Block Mode

SAI_MODEMASTER_TX

SAI_MODEMASTER_RX

SAI_MODESLAVE_TX

SAI_MODESLAVE_RX

SAI Block MSB LSB transmission

SAI_FIRSTBIT_MSB

SAI_FIRSTBIT_LSB

SAI Block Mute Value

SAI_ZERO_VALUE

SAI_LAST_SENT_VALUE

SAI Block NoDivider

SAI_MASTERDIVIDER_ENABLE

SAI_MASTERDIVIDER_DISABLE

SAI Block Output Drive

SAI_OUTPUTDRIVE_DISABLE

SAI_OUTPUTDRIVE_ENABLE

SAI Block Protocol

SAI_FREE_PROTOCOL

SAI_SPDIF_PROTOCOL

SAI_AC97_PROTOCOL

SAI Block Slot Active

SAI_SLOT_NOTACTIVE

SAI_SLOTACTIVE_0

SAI_SLOTACTIVE_1

SAI_SLOTACTIVE_2

SAI_SLOTACTIVE_3

SAI_SLOTACTIVE_4

SAI_SLOTACTIVE_5

SAI_SLOTACTIVE_6

SAI_SLOTACTIVE_7

SAI_SLOTACTIVE_8

SAI_SLOTACTIVE_9

SAI_SLOTACTIVE_10

SAI_SLOTACTIVE_11

SAI_SLOTACTIVE_12

SAI_SLOTACTIVE_13

SAI_SLOTACTIVE_14

SAI_SLOTACTIVE_15

SAI_SLOTACTIVE_ALL

SAI Block Slot Size

SAI_SLOTSIZE_DATASIZE

SAI_SLOTSIZE_16B

SAI_SLOTSIZE_32B

SAI External synchronisation

SAI_SYNCEXT_DISABLE

SAI_SYNCEXT_OUTBLOCKA_ENABLE

SAI_SYNCEXT_OUTBLOCKB_ENABLE

SAI Block Synchronization

SAI_ASYNCHRONOUS	Asynchronous
SAI_SYNCHRONOUS	Synchronous with other block of same SAI
SAI_SYNCHRONOUS_EXT_SAI1	Synchronous with other SAI, SAI1
SAI_SYNCHRONOUS_EXT_SAI2	Synchronous with other SAI, SAI2

SAI Error Code

HAL_SAI_ERROR_NONE	No error
HAL_SAI_ERROR_OVR	Overrun Error
HAL_SAI_ERROR_UDR	Underrun error
HAL_SAI_ERROR_AFSDET	Anticipated Frame synchronisation detection
HAL_SAI_ERROR_LFSDET	Late Frame synchronisation detection
HAL_SAI_ERROR_CNREADY	codec not ready
HAL_SAI_ERROR_WCKCFG	Wrong clock configuration
HAL_SAI_ERROR_TIMEOUT	Timeout error
HAL_SAI_ERROR_DMA	DMA error

SAI Exported Macros

__HAL_SAI_RESET_HANDLE_STATE

Description:

- Reset SAI handle state.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_ENABLE_IT

Description:

- Enable or disable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable

- SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
- SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_DISABLE_IT

__HAL_SAI_GET_IT_SOURCE

Description:

- Check whether the specified SAI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the SAI interrupt source to check. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_SAI_GET_FLAG

Description:

- Check whether the specified SAI flag is set or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SAI_FLAG_OVRUDR: Overrun underrun flag.
  - SAI_FLAG_MUTEDET: Mute detection flag.

- SAI_FLAG_WCKCFG: Wrong Clock Configuration flag.
- SAI_FLAG_FREQ: FIFO request flag.
- SAI_FLAG_CNRDY: Codec not ready flag.
- SAI_FLAG_AFSDET: Anticipated frame synchronization detection flag.
- SAI_FLAG_LFSDET: Late frame synchronization detection flag.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SAI_CLEAR_FLAG

Description:

- Clear the specified SAI pending flag.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SAI_FLAG_OVRUDR: Clear Overrun underrun
  - SAI_FLAG_MUTEDET: Clear Mute detection
  - SAI_FLAG_WCKCFG: Clear Wrong Clock Configuration
  - SAI_FLAG_FREQ: Clear FIFO request
  - SAI_FLAG_CNRDY: Clear Codec not ready
  - SAI_FLAG_AFSDET: Clear Anticipated frame synchronization detection
  - SAI_FLAG_LFSDET: Clear Late frame synchronization detection

Return value:

- None

__HAL_SAI_ENABLE

__HAL_SAI_DISABLE

SAI Mono Stereo Mode

SAI_STEREOMODE

SAI_MONOMODE

SAI PDM Clock Enable

SAI_PDM_CLOCK1_ENABLE

SAI_PDM_CLOCK2_ENABLE

SAI Supported protocol

SAI_I2S_STANDARD

SAI_I2S_MSBJUSTIFIED

SAI_I2S_LSBJUSTIFIED

SAI_PCM_LONG

SAI_PCM_SHORT

SAI protocol data size

SAI_PROTOCOL_DATASIZE_16BIT

SAI_PROTOCOL_DATASIZE_16BITEXTENDED

SAI_PROTOCOL_DATASIZE_24BIT

SAI_PROTOCOL_DATASIZE_32BIT

SAI TRIState Management

SAI_OUTPUT_NOTRELEASED

SAI_OUTPUT_RELEASED

57 HAL SAI Extension Driver

57.1 SAIEx Firmware driver registers structures

57.1.1 SAIEx_PdmMicDelayParamTypeDef

Data Fields

- ***uint32_t MicPair***
- ***uint32_t LeftDelay***
- ***uint32_t RightDelay***

Field Documentation

- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::MicPair***
  Specifies which pair of microphones is selected. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::LeftDelay***
  Specifies the delay in PDM clock unit to apply on left microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::RightDelay***
  Specifies the delay in PDM clock unit to apply on right microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.

57.2 SAIEx Firmware driver API description

57.2.1 Extended features functions

This section provides functions allowing to:

- Modify PDM microphone delays

This section contains the following APIs:

- ***HAL_SAIEx_ConfigPdmMicDelay()***

57.2.2 Detailed description of functions

HAL_SAIEx_ConfigPdmMicDelay

Function name	**HAL_StatusTypeDef HAL_SAIEx_ConfigPdmMicDelay (SAI_HandleTypeDef * hsai, SAIEx_PdmMicDelayParamTypeDef * pdmMicDelay)**
Function description	Configure PDM microphone delays.
Parameters	• **hsai:** SAI handle. • **pdmMicDelay:** Microphone delays configuration.
Return values	• **HAL:** status

58 HAL SD Extension Driver

58.1 SDEx Firmware driver API description

58.1.1 How to use this driver

The SD Extension HAL driver can be used as follows:

- Set card in High Speed mode using HAL_SDEx_HighSpeed() function.
- Configure Buffer0 and Buffer1 start address and Buffer size using HAL_SDEx_ConfigDMAMultiBuffer() function.
- Start Read and Write for multibuffer mode using HAL_SDEx_ReadBlocksDMAMultiBuffer() and HAL_SDEx_WriteBlocksDMAMultiBuffer() functions.

58.1.2 High Speed function

This section provides function allowing to configure the card in High Speed mode.

This section contains the following APIs:

- ***HAL_SDEx_HighSpeed()***
- ***HAL_SDEx_DriveTransceiver_1_8V_Callback()***

58.1.3 Multibuffer functions

This section provides functions allowing to configure the multibuffer mode and start read and write multibuffer mode for SD HAL driver.

This section contains the following APIs:

- ***HAL_SDEx_ConfigDMAMultiBuffer()***
- ***HAL_SDEx_ReadBlocksDMAMultiBuffer()***
- ***HAL_SDEx_WriteBlocksDMAMultiBuffer()***
- ***HAL_SDEx_ChangeDMABuffer()***
- ***HAL_SDEx_Read_DMADoubleBuffer0CpltCallback()***
- ***HAL_SDEx_Read_DMADoubleBuffer1CpltCallback()***
- ***HAL_SDEx_Write_DMADoubleBuffer0CpltCallback()***
- ***HAL_SDEx_Write_DMADoubleBuffer1CpltCallback()***

58.1.4 Detailed description of functions

HAL_SDEx_HighSpeed

Function name	**uint32_t HAL_SDEx_HighSpeed (SD_HandleTypeDef * hsd)**
Function description	Switches the SD card to High Speed mode.
Parameters	• **hsd:** SD handle
Return values	• **SD:** Card error state
Notes	• This operation should be followed by the configuration of PLL to have SDMMCCK clock between 50 and 120 MHz

HAL_SDEx_DriveTransceiver_1_8V_Callback

Function name	**void HAL_SDEx_DriveTransceiver_1_8V_Callback (FlagStatus status)**
Function description	Enable/Disable the SD Transciver 1.8V Mode Callback.
Parameters	• **status:** Voltage Switch State
Return values	• **None:**

HAL_SDEx_ConfigDMAMultiBuffer

Function name	**HAL_StatusTypeDef HAL_SDEx_ConfigDMAMultiBuffer (SD_HandleTypeDef * hsd, uint32_t * pDataBuffer0, uint32_t * pDataBuffer1, uint32_t BufferSize)**
Function description	Configure DMA Dual Buffer mode.
Parameters	• **hsd:** SD handle • **pDataBuffer0:** Pointer to the buffer0 that will contain/receive the transfered data • **pDataBuffer1:** Pointer to the buffer1 that will contain/receive the transfered data • **BufferSize:** Size of Buffer0 in Blocks. Buffer0 and Buffer1 must have the same size.
Return values	• **HAL:** status

HAL_SDEx_ReadBlocksDMAMultiBuffer

Function name	**HAL_StatusTypeDef HAL_SDEx_ReadBlocksDMAMultiBuffer (SD_HandleTypeDef * hsd, uint32_t BlockAdd, uint32_t NumberOfBlocks)**
Function description	Reads block(s) from a specified address in a card.
Parameters	• **hsd:** SD handle • **BlockAdd:** Block Address from where data is to be read • **NumberOfBlocks:** Total number of blocks to read
Return values	• **HAL:** status

HAL_SDEx_WriteBlocksDMAMultiBuffer

Function name	**HAL_StatusTypeDef HAL_SDEx_WriteBlocksDMAMultiBuffer (SD_HandleTypeDef * hsd, uint32_t BlockAdd, uint32_t NumberOfBlocks)**
Function description	Write block(s) to a specified address in a card.
Parameters	• **hsd:** SD handle • **BlockAdd:** Block Address from where data is to be read • **NumberOfBlocks:** Total number of blocks to read
Return values	• **HAL:** status

HAL_SDEx_ChangeDMABuffer

Function name	**HAL_StatusTypeDef HAL_SDEx_ChangeDMABuffer (SD_HandleTypeDef * hsd, HAL_SDEx_DMABuffer_MemoryTypeDef Buffer, uint32_t * pDataBuffer)**
Function description	Change the DMA Buffer0 or Buffer1 address on the fly.
Parameters	• **hsd:** pointer to a SD_HandleTypeDef structure. • **Buffer:** the buffer to be changed, This parameter can be one of the following values: SD_DMA_BUFFER0 or SD_DMA_BUFFER1 • **pDataBuffer:** The new address
Return values	• **HAL:** status
Notes	• The BUFFER0 address can be changed only when the current transfer use BUFFER1 and the BUFFER1 address can be changed only when the current transfer use BUFFER0.

HAL_SDEx_Read_DMADoubleBuffer0CpltCallback

Function name	**void HAL_SDEx_Read_DMADoubleBuffer0CpltCallback (SD_HandleTypeDef * hsd)**
Function description	Read DMA Buffer 0 Transfer completed callbacks.
Parameters	• **hsd:** SD handle
Return values	• **None:**

HAL_SDEx_Read_DMADoubleBuffer1CpltCallback

Function name	**void HAL_SDEx_Read_DMADoubleBuffer1CpltCallback (SD_HandleTypeDef * hsd)**
Function description	Read DMA Buffer 1 Transfer completed callbacks.
Parameters	• **hsd:** SD handle
Return values	• **None:**

HAL_SDEx_Write_DMADoubleBuffer0CpltCallback

Function name	**void HAL_SDEx_Write_DMADoubleBuffer0CpltCallback (SD_HandleTypeDef * hsd)**
Function description	Write DMA Buffer 0 Transfer completed callbacks.
Parameters	• **hsd:** SD handle
Return values	• **None:**

HAL_SDEx_Write_DMADoubleBuffer1CpltCallback

Function name	**void HAL_SDEx_Write_DMADoubleBuffer1CpltCallback (SD_HandleTypeDef * hsd)**
Function description	Write DMA Buffer 1 Transfer completed callbacks.
Parameters	• **hsd:** SD handle
Return values	• **None:**

59 HAL SMARTCARD Generic Driver

59.1 SMARTCARD Firmware driver registers structures

59.1.1 SMARTCARD_InitTypeDef

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint16_t Parity***
- ***uint16_t Mode***
- ***uint16_t CLKPolarity***
- ***uint16_t CLKPhase***
- ***uint16_t CLKLastBit***
- ***uint16_t OneBitSampling***
- ***uint8_t Prescaler***
- ***uint8_t GuardTime***
- ***uint16_t NACKEnable***
- ***uint32_t TimeOutEnable***
- ***uint32_t TimeOutValue***
- ***uint8_t BlockLength***
- ***uint8_t AutoRetryCount***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate***
  Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hsmartcard->Init.BaudRate))) where usart_ker_ckpres is the USART input clock divided by a prescaler
- ***uint32_t SMARTCARD_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter ***SMARTCARD_Word_Length*** can only be set to 9 (8 data + 1 parity bits).
- ***uint32_t SMARTCARD_InitTypeDef::StopBits***
  Specifies the number of stop bits. This parameter can be a value of ***SMARTCARD_Stop_Bits***.
- ***uint16_t SMARTCARD_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***SMARTCARD_Parity*** **Note:**The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.
- ***uint16_t SMARTCARD_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***SMARTCARD_Mode***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***SMARTCARD_Clock_Polarity***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***SMARTCARD_Clock_Phase***

- ***uint16_t SMARTCARD_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***SMARTCARD_Last_Bit***
- ***uint16_t SMARTCARD_InitTypeDef::OneBitSampling***
  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***SMARTCARD_OneBit_Sampling***.
- ***uint8_t SMARTCARD_InitTypeDef::Prescaler***
  Specifies the SmartCard Prescaler. This parameter can be any value from 0x01 to 0x1F. Prescaler value is multiplied by 2 to give the division factor of the source clock frequency
- ***uint8_t SMARTCARD_InitTypeDef::GuardTime***
  Specifies the SmartCard Guard Time applied after stop bits.
- ***uint16_t SMARTCARD_InitTypeDef::NACKEnable***
  Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of ***SMARTCARD_NACK_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutEnable***
  Specifies whether the receiver timeout is enabled. This parameter can be a value of ***SMARTCARD_Timeout_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutValue***
  Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- ***uint8_t SMARTCARD_InitTypeDef::BlockLength***
  Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- ***uint8_t SMARTCARD_InitTypeDef::AutoRetryCount***
  Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)
- ***uint32_t SMARTCARD_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***SMARTCARD_ClockPrescaler***.

59.1.2 SMARTCARD_AdvFeatureInitTypeDef

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t MSBFirst***
- ***uint16_t TxCompletionIndication***

Field Documentation

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit***
  Specifies which advanced SMARTCARD features is initialized. Several advanced features may be initialized at the same time. This parameter can be a value of ***SMARTCARDEx_Advanced_Features_Initialization_Type***

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert***
  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Tx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Rx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***SMARTCARD_Data_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***SMARTCARD_Rx_Tx_Swap***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***SMARTCARD_Overrun_Disable***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***SMARTCARD_DMA_Disable_on_Rx_Error***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***SMARTCARD_MSB_First***
- ***uint16_t SMARTCARD_AdvFeatureInitTypeDef::TxCompletionIndication***
  Specifies which transmission completion indication is used: before (when relevant flag is available) or once guard time period has elapsed. This parameter can be a value of ***SMARTCARDEx_Transmission_Completion_Indication***.

59.1.3 __SMARTCARD_HandleTypeDef

Data Fields

- ***USART_TypeDef * Instance***
- ***SMARTCARD_InitTypeDef Init***
- ***SMARTCARD_AdvFeatureInitTypeDef AdvancedInit***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t FifoMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef gState***
- ***__IO HAL_SMARTCARD_StateTypeDef RxState***
- ***uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* __SMARTCARD_HandleTypeDef::Instance***
  USART registers base address

- ***SMARTCARD_InitTypeDef __SMARTCARD_HandleTypeDef::Init***
  SmartCard communication parameters
- ***SMARTCARD_AdvFeatureInitTypeDef __SMARTCARD_HandleTypeDef::AdvancedInit***
  SmartCard advanced features initialization parameters
- ***uint8_t* __SMARTCARD_HandleTypeDef::pTxBuffPtr***
  Pointer to SmartCard Tx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::TxXferSize***
  SmartCard Tx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::TxXferCount***
  SmartCard Tx Transfer Counter
- ***uint8_t* __SMARTCARD_HandleTypeDef::pRxBuffPtr***
  Pointer to SmartCard Rx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::RxXferSize***
  SmartCard Rx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::RxXferCount***
  SmartCard Rx Transfer Counter
- ***uint16_t __SMARTCARD_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __SMARTCARD_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __SMARTCARD_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode is being used. This parameter can be a value of ***SMARTCARDEx_FIFO_mode***.
- ***void(* __SMARTCARD_HandleTypeDef::RxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __SMARTCARD_HandleTypeDef::TxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmatx***
  SmartCard Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmarx***
  SmartCard Rx DMA Handle parameters
- ***HAL_LockTypeDef __SMARTCARD_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::gState***
  SmartCard state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::RxState***
  SmartCard state information related to Rx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***uint32_t __SMARTCARD_HandleTypeDef::ErrorCode***
  SmartCard Error code

59.2 SMARTCARD Firmware driver API description

59.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure (eg. SMARTCARD_HandleTypeDef hsmartcard).
2. Associate a USART to the SMARTCARD handle hsmartcard.

3. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
   - Enable the USARTx interface clock.
   - USART pins configuration:
     - Enable the clock for the USART GPIOs.
     - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
   - NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
4. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsmartcard handle Init structure.
5. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard handle AdvancedInit structure.
6. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit() API.

The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_SMARTCARD_ENABLE_IT() and __HAL_SMARTCARD_DISABLE_IT() inside the transmit and receive process.

Three operation modes are available within this driver:

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SMARTCARD_Transmit()
- Receive an amount of data in blocking mode using HAL_SMARTCARD_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SMARTCARD_Transmit_IT()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()

- Receive an amount of data in non-blocking mode using HAL_SMARTCARD_Receive_IT()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Transmit_DMA()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Receive_DMA()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

SMARTCARD HAL driver macros list

Below the list of most used macros in SMARTCARD HAL driver.

- __HAL_SMARTCARD_GET_FLAG: Check whether or not the specified SMARTCARD flag is set
- __HAL_SMARTCARD_CLEAR_FLAG: Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_GET_IT_SOURCE: Check whether or not the specified SMARTCARD interrupt is enabled

You can refer to the SMARTCARD HAL driver header file for more useful macros

59.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

- These parameters can be configured:
  - Baud Rate
  - Parity: parity should be enabled, frame Length is fixed to 8 bits plus parity
  - Receiver/transmitter modes
  - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
  - Prescaler value
  - Guard bit time
  - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:

  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - Time out enabling (and if activated, timeout value)
  - Block length
  - Auto-retry counter

The HAL_SMARTCARD_Init() API follows the USART synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_SMARTCARD_Init()***
- ***HAL_SMARTCARD_DeInit()***
- ***HAL_SMARTCARD_MspInit()***
- ***HAL_SMARTCARD_MspDeInit()***

59.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SMARTCARD data transfers.

Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as:

- 8 bits plus parity: where M=1 and PCE=1 in the USART_CR1 register
- 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART_CR2 register.

There are two modes of transfer:

- Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
- Non-blocking mode: The communication is performed using Interrupts or DMA, the relevant APIs return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.

The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected.

In Blocking mode, the APIs are:

- HAL_SMARTCARD_Transmit()
- HAL_SMARTCARD_Receive()

In Non-blocking mode, the APIs with Interrupt are:

- HAL_SMARTCARD_Transmit_IT()
- HAL_SMARTCARD_Receive_IT()
- HAL_SMARTCARD_IRQHandler()

In Non-blocking mode, the functions with DMA are:

- HAL_SMARTCARD_Transmit_DMA()
- HAL_SMARTCARD_Receive_DMA()

A set of Transfer Complete Callbacks are provided in Non-blocking mode:

- HAL_SMARTCARD_TxCpltCallback()
- HAL_SMARTCARD_RxCpltCallback()
- HAL_SMARTCARD_ErrorCallback()

1. Non-Blocking mode transfers could be aborted using Abort APIs:
   - HAL_SMARTCARD_Abort()
   - HAL_SMARTCARD_AbortTransmit()
   - HAL_SMARTCARD_AbortReceive()
   - HAL_SMARTCARD_Abort_IT()
   - HAL_SMARTCARD_AbortTransmit_IT()
   - HAL_SMARTCARD_AbortReceive_IT()
2. For Abort services based on interrupts (HAL_SMARTCARD_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
   - HAL_SMARTCARD_AbortCpltCallback()
   - HAL_SMARTCARD_AbortTransmitCpltCallback()
   - HAL_SMARTCARD_AbortReceiveCpltCallback()
3. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows:
   - Error is considered as Recoverable and non blocking: Transfer could go till end, but error severity is to be evaluated by user: this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed. Transfer is kept ongoing on SMARTCARD side. If user wants to abort it, Abort services should be called by user.
   - Error is considered as Blocking: Transfer could not be completed properly and is aborted. This concerns Frame Error in Interrupt mode tranmission, Overrun Error in Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed.

- There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - Non-Blocking mode: The communication is performed using Interrupts or DMA, the relevant APIs return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
  - The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected.
- Blocking mode APIs are:
  - HAL_SMARTCARD_Transmit()
  - HAL_SMARTCARD_Receive()
- Non Blocking mode APIs with Interrupt are:
  - HAL_SMARTCARD_Transmit_IT()

  - HAL_SMARTCARD_Receive_IT()
  - HAL_SMARTCARD_IRQHandler()
- Non Blocking mode functions with DMA are:
  - HAL_SMARTCARD_Transmit_DMA()
  - HAL_SMARTCARD_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL_SMARTCARD_TxCpltCallback()
  - HAL_SMARTCARD_RxCpltCallback()
  - HAL_SMARTCARD_ErrorCallback() (#) Non-Blocking mode transfers could be aborted using Abort APIs:
- HAL_SMARTCARD_Abort()
- HAL_SMARTCARD_AbortTransmit()
- HAL_SMARTCARD_AbortReceive()
- HAL_SMARTCARD_Abort_IT()
- HAL_SMARTCARD_AbortTransmit_IT()
- HAL_SMARTCARD_AbortReceive_IT() (#) For Abort services based on interrupts (HAL_SMARTCARD_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
- HAL_SMARTCARD_AbortCpltCallback()
- HAL_SMARTCARD_AbortTransmitCpltCallback()
- HAL_SMARTCARD_AbortReceiveCpltCallback() (#) In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows:
- Error is considered as Recoverable and non blocking: Transfer could go till end, but error severity is to be evaluated by user: this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed. Transfer is kept ongoing on SMARTCARD side. If user wants to abort it, Abort services should be called by user.
- Error is considered as Blocking: Transfer could not be completed properly and is aborted. This concerns Frame Error in Interrupt mode tranmission, Overrun Error in Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_SMARTCARD_Transmit()***
- ***HAL_SMARTCARD_Receive()***
- ***HAL_SMARTCARD_Transmit_IT()***
- ***HAL_SMARTCARD_Receive_IT()***
- ***HAL_SMARTCARD_Transmit_DMA()***
- ***HAL_SMARTCARD_Receive_DMA()***
- ***HAL_SMARTCARD_Abort()***
- ***HAL_SMARTCARD_AbortTransmit()***
- ***HAL_SMARTCARD_AbortReceive()***
- ***HAL_SMARTCARD_Abort_IT()***
- ***HAL_SMARTCARD_AbortTransmit_IT()***
- ***HAL_SMARTCARD_AbortReceive_IT()***
- ***HAL_SMARTCARD_IRQHandler()***
- ***HAL_SMARTCARD_TxCpltCallback()***
- ***HAL_SMARTCARD_RxCpltCallback()***
- ***HAL_SMARTCARD_ErrorCallback()***
- ***HAL_SMARTCARD_AbortCpltCallback()***
- ***HAL_SMARTCARD_AbortTransmitCpltCallback()***
- ***HAL_SMARTCARD_AbortReceiveCpltCallback()***

59.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of SmartCard handle and also return Peripheral Errors occurred during communication process

- HAL_SMARTCARD_GetState() API can be helpful to check in run-time the state of the SMARTCARD peripheral.
- HAL_SMARTCARD_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- ***HAL_SMARTCARD_GetState()***
- ***HAL_SMARTCARD_GetError()***

59.2.5 Detailed description of functions

HAL_SMARTCARD_Init

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Initialize the SMARTCARD mode according to the specified parameters in the SMARTCARD_HandleTypeDef and initialize the associated handle.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status

HAL_SMARTCARD_DeInit

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	DeInitialize the SMARTCARD peripheral.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status

HAL_SMARTCARD_MspInit

Function name	**void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Initialize the SMARTCARD MSP.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None:**

HAL_SMARTCARD_MspDeInit

Function name	**void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	DeInitialize the SMARTCARD MSP.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None:**

HAL_SMARTCARD_Transmit

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Send an amount of data in blocking mode.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **pData:** pointer to data buffer. • **Size:** amount of data to be sent. • **Timeout:** Timeout duration.
Return values	• **HAL:** status
Notes	• When FIFO mode is enabled, writing a data in the TDR register adds one data to the TXFIFO. Write operations to the TDR register are performed when TXFNF flag is set. From hardware perspective, TXFNF flag and TXE are mapped on the same bit-field.

HAL_SMARTCARD_Receive

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **pData:** pointer to data buffer. • **Size:** amount of data to be received. • **Timeout:** Timeout duration.
Return values	• **HAL:** status
Notes	• When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_SMARTCARD_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)**
Function description	Send an amount of data in interrupt mode.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **pData:** pointer to data buffer. • **Size:** amount of data to be sent.
Return values	• **HAL:** status
Notes	• When FIFO mode is disabled, USART interrupt is generated whenever USART_TDR register is empty, i.e one interrupt per data to transmit. • When FIFO mode is enabled, USART interrupt is generated whenever TXFIFO threshold reached. In that case the interrupt rate depends on TXFIFO threshold configuration. • This function sets the hsmartcard->TxIsr function pointer according to the FIFO mode (data transmission processing depends on FIFO mode).

HAL_SMARTCARD_Receive_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in interrupt mode.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **pData:** pointer to data buffer. • **Size:** amount of data to be received.
Return values	• **HAL:** status
Notes	• When FIFO mode is disabled, USART interrupt is generated whenever USART_RDR register can be read, i.e one interrupt per data to receive. • When FIFO mode is enabled, USART interrupt is generated whenever RXFIFO threshold reached. In that case the interrupt rate depends on RXFIFO threshold configuration. • This function sets the hsmartcard->RxIsr function pointer according to the FIFO mode (data reception processing depends on FIFO mode).

HAL_SMARTCARD_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)**

Function description	Send an amount of data in DMA mode.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **pData:** pointer to data buffer. • **Size:** amount of data to be sent.
Return values	• **HAL:** status

HAL_SMARTCARD_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in DMA mode.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **pData:** pointer to data buffer. • **Size:** amount of data to be received.
Return values	• **HAL:** status
Notes	• The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).

HAL_SMARTCARD_Abort

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Abort (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Abort ongoing transfers (blocking mode).
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode: when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortTransmit

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Abort ongoing Transmit transfer (blocking mode).

Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode: when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortReceive

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Abort ongoing Receive transfer (blocking mode).
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode: when exiting function, Abort is considered as completed.

HAL_SMARTCARD_Abort_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Abort_IT (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Abort ongoing transfers (Interrupt mode).
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort

complete callback

- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortTransmit_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit_IT (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Abort ongoing Transmit transfer (Interrupt mode).
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortReceive_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive_IT (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Abort ongoing Receive transfer (Interrupt mode).
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_IRQHandler

Function name	**void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Handle SMARTCARD interrupt requests.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None:**

HAL_SMARTCARD_TxCpltCallback

Function name	**void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Tx Transfer completed callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None:**

HAL_SMARTCARD_RxCpltCallback

Function name	**void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Rx Transfer completed callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None:**

HAL_SMARTCARD_ErrorCallback

Function name	**void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	SMARTCARD error callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None:**

HAL_SMARTCARD_AbortCpltCallback

Function name	**void HAL_SMARTCARD_AbortCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	SMARTCARD Abort Complete callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the

specified SMARTCARD module.

Return values	• **None:**

HAL_SMARTCARD_AbortTransmitCpltCallback

Function name	**void HAL_SMARTCARD_AbortTransmitCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	SMARTCARD Abort Complete callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None:**

HAL_SMARTCARD_AbortReceiveCpltCallback

Function name	**void HAL_SMARTCARD_AbortReceiveCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	SMARTCARD Abort Receive Complete callback.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **None:**

HAL_SMARTCARD_GetState

Function name	**HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Return the SMARTCARD handle state.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **SMARTCARD:** handle state

HAL_SMARTCARD_GetError

Function name	**uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Return the SMARTCARD handle error code.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **SMARTCARD:** handle Error Code

59.3 SMARTCARD Firmware driver defines

59.3.1 SMARTCARD

Clock Prescaler

SMARTCARD_PRESCALER_DIV1	fclk_pres = fclk
SMARTCARD_PRESCALER_DIV2	fclk_pres = fclk/2
SMARTCARD_PRESCALER_DIV4	fclk_pres = fclk/4
SMARTCARD_PRESCALER_DIV6	fclk_pres = fclk/6
SMARTCARD_PRESCALER_DIV8	fclk_pres = fclk/8
SMARTCARD_PRESCALER_DIV10	fclk_pres = fclk/10
SMARTCARD_PRESCALER_DIV12	fclk_pres = fclk/12
SMARTCARD_PRESCALER_DIV16	fclk_pres = fclk/16
SMARTCARD_PRESCALER_DIV32	fclk_pres = fclk/32
SMARTCARD_PRESCALER_DIV64	fclk_pres = fclk/64
SMARTCARD_PRESCALER_DIV128	fclk_pres = fclk/128
SMARTCARD_PRESCALER_DIV256	fclk_pres = fclk/256

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE	SMARTCARD frame phase on first clock transition
SMARTCARD_PHASE_2EDGE	SMARTCARD frame phase on second clock transition

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW	SMARTCARD frame low polarity
SMARTCARD_POLARITY_HIGH	SMARTCARD frame high polarity

SMARTCARD advanced feature Binary Data inversion

SMARTCARD_ADVFEATURE_DATAINV_DISABLE	Binary data inversion disable
SMARTCARD_ADVFEATURE_DATAINV_ENABLE	Binary data inversion enable

SMARTCARD advanced feature DMA Disable on Rx Error

SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR	DMA enable on Reception Error
SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR	DMA disable on Reception Error

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:

- Reset SMARTCARD handle states.

Parameters:

- __HANDLE__: SMARTCARD handle.

Return value:

- None

__HAL_SMARTCARD_FLUSH_DRREGISTER

Description:

- Flush the Smartcard Data registers.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FLAG

Description:

- Clear the specified SMARTCARD pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detected clear flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear flag

Return value:

- None

__HAL_SMARTCARD_CLEAR_PEFLAG

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FE FLAG	**Description:** • Clear the SMARTCARD FE pending flag. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. **Return value:** • None
__HAL_SMARTCARD_CLEAR_NE FLAG	**Description:** • Clear the SMARTCARD NE pending flag. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. **Return value:** • None
__HAL_SMARTCARD_CLEAR_OR E FLAG	**Description:** • Clear the SMARTCARD ORE pending flag. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. **Return value:** • None
__HAL_SMARTCARD_CLEAR_ IDLE FLAG	**Description:** • Clear the SMARTCARD IDLE pending flag. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. **Return value:** • None
__HAL_SMARTCARD_GET_FLAG	**Description:** • Check whether the specified Smartcard flag is set or not. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. • __FLAG__: specifies the flag to check. This parameter can be one of the following values: – SMARTCARD_FLAG_TCBGT Transmission complete before guard time flag (when flag available) – SMARTCARD_FLAG_REACK Receive

enable acknowledge flag
 – SMARTCARD_FLAG_TEACK Transmit enable acknowledge flag
 – SMARTCARD_FLAG_BUSY Busy flag
 – SMARTCARD_FLAG_EOBF End of block flag
 – SMARTCARD_FLAG_RTOF Receiver timeout flag
 – SMARTCARD_FLAG_TXE Transmit data register empty flag
 – SMARTCARD_FLAG_TXFNF TXFIFO not full flag
 – SMARTCARD_FLAG_TC Transmission complete flag
 – SMARTCARD_FLAG_RXNE Receive data register not empty flag
 – SMARTCARD_FLAG_RXFNE RXFIFO not empty flag
 – SMARTCARD_FLAG_IDLE Idle line detection flag
 – SMARTCARD_FLAG_ORE Overrun error flag
 – SMARTCARD_FLAG_NE Noise error flag
 – SMARTCARD_FLAG_FE Framing error flag
 – SMARTCARD_FLAG_PE Parity error flag
 – SMARTCARD_FLAG_TXFE TXFIFO Empty flag
 – SMARTCARD_FLAG_RXFF RXFIFO Full flag
 – SMARTCARD_FLAG_RXFT SMARTCARD RXFIFO threshold flag
 – SMARTCARD_FLAG_TXFT SMARTCARD TXFIFO threshold flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SMARTCARD_ENABLE_IT

Description:

- Enable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  – SMARTCARD_IT_EOB End of block interrupt
  – SMARTCARD_IT_RTO Receive timeout interrupt
  – SMARTCARD_IT_TXE Transmit data register empty interrupt

- SMARTCARD_IT_TXFNF TX FIFO not full interruption
- SMARTCARD_IT_TC Transmission complete interrupt
- SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
- SMARTCARD_IT_RXNE Receive data register not empty interrupt
- SMARTCARD_IT_RXFNE RXFIFO not empty interruption
- SMARTCARD_IT_IDLE Idle line detection interrupt
- SMARTCARD_IT_PE Parity error interrupt
- SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
- SMARTCARD_IT_RXFF RXFIFO full interruption
- SMARTCARD_IT_TXFE TXFIFO empty interruption
- SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
- SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_DISABLE_IT

Description:

- Disable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_RXFNE RXFIFO not

empty interruption
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_GET_IT

Description:

- Check whether the specified SmartCard interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_RXFF RXFIFO full interruption

- SMARTCARD_IT_TXFE TXFIFO empty interruption
- SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
- SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_GET_IT_SOURCE

Description:

- Check whether the specified SmartCard interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_CLEAR_IT

Description:

- Clear the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detection clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear Flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag (when flag available)
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

__HAL_SMARTCARD_SEND_REQ

Description:

- Set a specific SMARTCARD request flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - SMARTCARD_RXDATA_FLUSH_REQUEST Receive data flush Request
  - SMARTCARD_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE	**Description:** • Enable the SMARTCARD one bit sample method. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. **Return value:** • None
__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE	**Description:** • Disable the SMARTCARD one bit sample method. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. **Return value:** • None
__HAL_SMARTCARD_ENABLE	**Description:** • Enable the USART associated to the SMARTCARD Handle. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. **Return value:** • None
__HAL_SMARTCARD_DISABLE	**Description:** • Disable the USART associated to the SMARTCARD Handle. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. **Return value:** • None

SMARTCARD interruptions flags mask

SMARTCARD_IT_MASK	SMARTCARD interruptions flags mask

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE	SMARTCARD frame last data bit clock pulse not output to SCLK pin
SMARTCARD_LASTBIT_ENABLE	SMARTCARD frame last data bit clock pulse output

to SCLK pin

SMARTCARD Transfer Mode

SMARTCARD_MODE_RX	SMARTCARD RX mode
SMARTCARD_MODE_TX	SMARTCARD TX mode
SMARTCARD_MODE_TX_RX	SMARTCARD RX and TX mode

SMARTCARD advanced feature MSB first

SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE	Most significant bit sent/received first disable
SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE	Most significant bit sent/received first enable

SMARTCARD NACK Enable

SMARTCARD_NACK_DISABLE	SMARTCARD NACK transmission disabled
SMARTCARD_NACK_ENABLE	SMARTCARD NACK transmission enabled

SMARTCARD One Bit Sampling Method

SMARTCARD_ONE_BIT_SAMPLE_DISABLE	SMARTCARD frame one-bit sample disabled
SMARTCARD_ONE_BIT_SAMPLE_ENABLE	SMARTCARD frame one-bit sample enabled

SMARTCARD advanced feature Overrun Disable

SMARTCARD_ADVFEATURE_OVERRUN_ENABLE	RX overrun enable
SMARTCARD_ADVFEATURE_OVERRUN_DISABLE	RX overrun disable

SMARTCARD Parity

SMARTCARD_PARITY_EVEN	SMARTCARD frame even parity
SMARTCARD_PARITY_ODD	SMARTCARD frame odd parity

SMARTCARD Request Parameters

SMARTCARD_RXDATA_FLUSH_REQUEST	Receive data flush request
SMARTCARD_TXDATA_FLUSH_REQUEST	Transmit data flush request

SMARTCARD advanced feature RX pin active level inversion

SMARTCARD_ADVFEATURE_RXINV_DISABLE	RX pin active level inversion disable
SMARTCARD_ADVFEATURE_RXINV_ENABLE	RX pin active level inversion enable

SMARTCARD advanced feature RX TX pins swap

SMARTCARD_ADVFEATURE_SWAP_DISABLE	TX/RX pins swap disable
SMARTCARD_ADVFEATURE_SWAP_ENABLE	TX/RX pins swap enable

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0_5	SMARTCARD frame with 0.5 stop bit
SMARTCARD_STOPBITS_1_5	SMARTCARD frame with 1.5 stop bits

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DISABLE	SMARTCARD receiver timeout disabled

SMARTCARD_TIMEOUT_ENABLE	SMARTCARD receiver timeout enabled

SMARTCARD advanced feature TX pin active level inversion

SMARTCARD_ADVFEATURE_TXINV_DISABLE	TX pin active level inversion disable
SMARTCARD_ADVFEATURE_TXINV_ENABLE	TX pin active level inversion enable

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B	SMARTCARD frame length

60 HAL SMARTCARD Extension Driver

60.1 SMARTCARDEx Firmware driver API description

60.1.1 SMARTCARD peripheral extended features

The Extended SMARTCARD HAL driver can be used as follows:

1. After having configured the SMARTCARD basic features with HAL_SMARTCARD_Init(), then program SMARTCARD advanced features if required (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard AdvancedInit structure.

60.1.2 IO operation functions

This section contains the following APIs:

- *HAL_SMARTCARDEx_RxFifoFullCallback()*
- *HAL_SMARTCARDEx_TxFifoEmptyCallback()*

60.1.3 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL_SMARTCARDEx_BlockLength_Config() API allows to configure the Block Length on the fly
- HAL_SMARTCARDEx_TimeOut_Config() API allows to configure the receiver timeout value on the fly
- HAL_SMARTCARDEx_EnableReceiverTimeOut() API enables the receiver timeout feature
- HAL_SMARTCARDEx_DisableReceiverTimeOut() API disables the receiver timeout feature
- HAL_SMARTCARDEx_EnableFifoMode() API enables the FIFO mode
- HAL_SMARTCARDEx_DisableFifoMode() API disables the FIFO mode
- HAL_SMARTCARDEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_SMARTCARDEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- *HAL_SMARTCARDEx_BlockLength_Config()*
- *HAL_SMARTCARDEx_TimeOut_Config()*
- *HAL_SMARTCARDEx_EnableReceiverTimeOut()*
- *HAL_SMARTCARDEx_DisableReceiverTimeOut()*
- *HAL_SMARTCARDEx_EnableFifoMode()*
- *HAL_SMARTCARDEx_DisableFifoMode()*
- *HAL_SMARTCARDEx_SetTxFifoThreshold()*
- *HAL_SMARTCARDEx_SetRxFifoThreshold()*

60.1.4 Detailed description of functions

HAL_SMARTCARDEx_RxFifoFullCallback

Function name	**void HAL_SMARTCARDEx_RxFifoFullCallback (SMARTCARD_HandleTypeDef * hsmartcard)**

Function description	SMARTCARD RX Fifo full callback.
Parameters	• **hsmartcard:** SMARTCARD handle.
Return values	• **None:**

HAL_SMARTCARDEx_TxFifoEmptyCallback

Function name	**void HAL_SMARTCARDEx_TxFifoEmptyCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	SMARTCARD TX Fifo empty callback.
Parameters	• **hsmartcard:** SMARTCARD handle.
Return values	• **None:**

HAL_SMARTCARDEx_BlockLength_Config

Function name	**void HAL_SMARTCARDEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t BlockLength)**
Function description	Update on the fly the SMARTCARD block length in RTOR register.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **BlockLength:** SMARTCARD block length (8-bit long at most)
Return values	• **None:**

HAL_SMARTCARDEx_TimeOut_Config

Function name	**void HAL_SMARTCARDEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t TimeOutValue)**
Function description	Update on the fly the receiver timeout value in RTOR register.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **TimeOutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFFF.
Return values	• **None:**

HAL_SMARTCARDEx_EnableReceiverTimeOut

Function name	**HAL_StatusTypeDef HAL_SMARTCARDEx_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Enable the SMARTCARD receiver timeout feature.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the

specified SMARTCARD module.

Return values
- **HAL:** status

HAL_SMARTCARDEx_DisableReceiverTimeOut

Function name	**HAL_StatusTypeDef HAL_SMARTCARDEx_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Disable the SMARTCARD receiver timeout feature.
Parameters	• **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• **HAL:** status

HAL_SMARTCARDEx_EnableFifoMode

Function name	**HAL_StatusTypeDef HAL_SMARTCARDEx_EnableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Enable the FIFO mode.
Parameters	• **hsmartcard:** SMARTCARD handle.
Return values	• **HAL:** status

HAL_SMARTCARDEx_DisableFifoMode

Function name	**HAL_StatusTypeDef HAL_SMARTCARDEx_DisableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)**
Function description	Disable the FIFO mode.
Parameters	• **hsmartcard:** SMARTCARD handle.
Return values	• **HAL:** status

HAL_SMARTCARDEx_SetTxFifoThreshold

Function name	**HAL_StatusTypeDef HAL_SMARTCARDEx_SetTxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)**
Function description	Set the TXFIFO threshold.
Parameters	• **hsmartcard:** SMARTCARD handle. • **Threshold:** TX FIFO threshold value This parameter can be one of the following values: – SMARTCARD_TXFIFO_THRESHOLD_1_8 – SMARTCARD_TXFIFO_THRESHOLD_1_4 – SMARTCARD_TXFIFO_THRESHOLD_1_2 – SMARTCARD_TXFIFO_THRESHOLD_3_4 – SMARTCARD_TXFIFO_THRESHOLD_7_8 – SMARTCARD_TXFIFO_THRESHOLD_8_8

Return values	• **HAL:** status

HAL_SMARTCARDEx_SetRxFifoThreshold

Function name	**HAL_StatusTypeDef HAL_SMARTCARDEx_SetRxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)**
Function description	Set the RXFIFO threshold.
Parameters	• **hsmartcard:** SMARTCARD handle. • **Threshold:** RX FIFO threshold value This parameter can be one of the following values: – SMARTCARD_RXFIFO_THRESHOLD_1_8 – SMARTCARD_RXFIFO_THRESHOLD_1_4 – SMARTCARD_RXFIFO_THRESHOLD_1_2 – SMARTCARD_RXFIFO_THRESHOLD_3_4 – SMARTCARD_RXFIFO_THRESHOLD_7_8 – SMARTCARD_RXFIFO_THRESHOLD_8_8
Return values	• **HAL:** status

60.2 SMARTCARDEx Firmware driver defines

60.2.1 SMARTCARDEx

SMARTCARD advanced feature initialization type

SMARTCARD_ADVFEATURE_NO_INIT	No advanced feature initialization
SMARTCARD_ADVFEATURE_TXINVERT_INIT	TX pin active level inversion
SMARTCARD_ADVFEATURE_RXINVERT_INIT	RX pin active level inversion
SMARTCARD_ADVFEATURE_DATAINVERT_INIT	Binary data inversion
SMARTCARD_ADVFEATURE_SWAP_INIT	TX/RX pins swap
SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT	RX overrun disable
SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT	DMA disable on Reception Error
SMARTCARD_ADVFEATURE_MSBFIRST_INIT	Most significant bit sent/received first
SMARTCARD_ADVFEATURE_TXCOMPLETION	TX completion indication before of after guard time

SMARTCARDEx FIFO mode

SMARTCARD_FIFOMODE_DISABLE	FIFO mode disable
SMARTCARD_FIFOMODE_ENABLE	FIFO mode enable

SMARTCARD Flags

SMARTCARD_FLAG_TCBGT	SMARTCARD transmission complete before guard time completion
SMARTCARD_FLAG_REACK	SMARTCARD receive enable acknowledge flag
SMARTCARD_FLAG_TEACK	SMARTCARD transmit enable acknowledge flag
SMARTCARD_FLAG_BUSY	SMARTCARD busy flag
SMARTCARD_FLAG_EOBF	SMARTCARD end of block flag
SMARTCARD_FLAG_RTOF	SMARTCARD receiver timeout flag
SMARTCARD_FLAG_TXE	SMARTCARD transmit data register empty
SMARTCARD_FLAG_TXFNF	SMARTCARD TXFIFO not full
SMARTCARD_FLAG_TC	SMARTCARD transmission complete
SMARTCARD_FLAG_RXNE	SMARTCARD read data register not empty
SMARTCARD_FLAG_RXFNE	SMARTCARD RXFIFO not empty
SMARTCARD_FLAG_IDLE	SMARTCARD idle line detection
SMARTCARD_FLAG_ORE	SMARTCARD overrun error
SMARTCARD_FLAG_NE	SMARTCARD noise error
SMARTCARD_FLAG_FE	SMARTCARD frame error
SMARTCARD_FLAG_PE	SMARTCARD parity error
SMARTCARD_FLAG_TXFE	SMARTCARD TXFIFO Empty flag
SMARTCARD_FLAG_RXFF	SMARTCARD RXFIFO Full flag
SMARTCARD_FLAG_RXFT	SMARTCARD RXFIFO threshold flag
SMARTCARD_FLAG_TXFT	SMARTCARD TXFIFO threshold flag

SMARTCARD Interrupts Definition

SMARTCARD_IT_PE	SMARTCARD parity error interruption
SMARTCARD_IT_TXE	SMARTCARD transmit data register empty interruption
SMARTCARD_IT_TXFNF	SMARTCARD TX FIFO not full interruption
SMARTCARD_IT_TC	SMARTCARD transmission complete interruption
SMARTCARD_IT_RXNE	SMARTCARD read data register not empty interruption
SMARTCARD_IT_RXFNE	SMARTCARD RXFIFO not empty interruption
SMARTCARD_IT_IDLE	SMARTCARD idle line detection interruption
SMARTCARD_IT_ERR	SMARTCARD error interruption
SMARTCARD_IT_ORE	SMARTCARD overrun error interruption
SMARTCARD_IT_NE	SMARTCARD noise error interruption
SMARTCARD_IT_FE	SMARTCARD frame error interruption
SMARTCARD_IT_EOB	SMARTCARD end of block interruption
SMARTCARD_IT_RTO	SMARTCARD receiver timeout interruption
SMARTCARD_IT_TCBGT	SMARTCARD transmission complete before guard time completion interruption

SMARTCARD_IT_RXFF	SMARTCARD RXFIFO full interruption
SMARTCARD_IT_TXFE	SMARTCARD TXFIFO empty interruption
SMARTCARD_IT_RXFT	SMARTCARD RXFIFO threshold reached interruption
SMARTCARD_IT_TXFT	SMARTCARD TXFIFO threshold reached interruption

SMARTCARD Interruption Clear Flags

SMARTCARD_CLEAR_PEF	SMARTCARD parity error clear flag
SMARTCARD_CLEAR_FEF	SMARTCARD framing error clear flag
SMARTCARD_CLEAR_NEF	SMARTCARD noise detected clear flag
SMARTCARD_CLEAR_OREF	SMARTCARD overrun error clear flag
SMARTCARD_CLEAR_IDLEF	SMARTCARD idle line detected clear flag
SMARTCARD_CLEAR_TXFECF	TXFIFO empty Clear Flag
SMARTCARD_CLEAR_TCF	SMARTCARD transmission complete clear flag
SMARTCARD_CLEAR_TCBGTF	SMARTCARD transmission complete before guard time completion clear flag
SMARTCARD_CLEAR_RTOF	SMARTCARD receiver time out clear flag
SMARTCARD_CLEAR_EOBF	SMARTCARD end of block clear flag

SMARTCARDEx RXFIFO threshold level

SMARTCARD_RXFIFO_THRESHOLD_1_8	RXFIFO FIFO reaches 1/8 of its depth
SMARTCARD_RXFIFO_THRESHOLD_1_4	RXFIFO FIFO reaches 1/4 of its depth
SMARTCARD_RXFIFO_THRESHOLD_1_2	RXFIFO FIFO reaches 1/2 of its depth
SMARTCARD_RXFIFO_THRESHOLD_3_4	RXFIFO FIFO reaches 3/4 of its depth
SMARTCARD_RXFIFO_THRESHOLD_7_8	RXFIFO FIFO reaches 7/8 of its depth
SMARTCARD_RXFIFO_THRESHOLD_8_8	RXFIFO FIFO becomes full

SMARTCARD Transmission Completion Indication

SMARTCARD_TCBGT	SMARTCARD transmission complete before guard time
SMARTCARD_TC	SMARTCARD transmission complete (flag raised when guard time has elapsed)

SMARTCARDEx TXFIFO threshold level

SMARTCARD_TXFIFO_THRESHOLD_1_8	TXFIFO reaches 1/8 of its depth
SMARTCARD_TXFIFO_THRESHOLD_1_4	TXFIFO reaches 1/4 of its depth
SMARTCARD_TXFIFO_THRESHOLD_1_2	TXFIFO reaches 1/2 of its depth
SMARTCARD_TXFIFO_THRESHOLD_3_4	TXFIFO reaches 3/4 of its depth
SMARTCARD_TXFIFO_THRESHOLD_7_8	TXFIFO reaches 7/8 of its depth
SMARTCARD_TXFIFO_THRESHOLD_8_8	TXFIFO becomes empty

61 HAL SMBUS Generic Driver

61.1 SMBUS Firmware driver registers structures

61.1.1 SMBUS_InitTypeDef

Data Fields

- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***
- ***uint32_t PacketErrorCheckMode***
- ***uint32_t PeripheralMode***
- ***uint32_t SMBusTimeout***

Field Documentation

- ***uint32_t SMBUS_InitTypeDef::Timing***
  Specifies the SMBUS_TIMINGR_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual
- ***uint32_t SMBUS_InitTypeDef::AnalogFilter***
  Specifies if Analog Filter is enable or not. This parameter can be a value of ***SMBUS_Analog_Filter***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t SMBUS_InitTypeDef::AddressingMode***
  Specifies if 7-bit or 10-bit addressing mode for master is selected. This parameter can be a value of ***SMBUS_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***SMBUS_dual_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2Masks***
  Specifies the acknoledge mask address second device own address if dual addressing mode is selected This parameter can be a value of ***SMBUS_own_address2_masks***.
- ***uint32_t SMBUS_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***SMBUS_general_call_addressing_mode***.
- ***uint32_t SMBUS_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***SMBUS_nostretch_mode***

- ***uint32_t SMBUS_InitTypeDef::PacketErrorCheckMode***
  Specifies if Packet Error Check mode is selected. This parameter can be a value of ***SMBUS_packet_error_check_mode***
- ***uint32_t SMBUS_InitTypeDef::PeripheralMode***
  Specifies which mode of Periphal is selected. This parameter can be a value of ***SMBUS_peripheral_mode***
- ***uint32_t SMBUS_InitTypeDef::SMBusTimeout***
  Specifies the content of the 32 Bits SMBUS_TIMEOUT_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

61.1.2 SMBUS_HandleTypeDef

Data Fields

- ***I2C_TypeDef * Instance***
- ***SMBUS_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***I2C_TypeDef* SMBUS_HandleTypeDef::Instance***
  SMBUS registers base address
- ***SMBUS_InitTypeDef SMBUS_HandleTypeDef::Init***
  SMBUS communication parameters
- ***uint8_t* SMBUS_HandleTypeDef::pBuffPtr***
  Pointer to SMBUS transfer buffer
- ***uint16_t SMBUS_HandleTypeDef::XferSize***
  SMBUS transfer size
- ***__IO uint16_t SMBUS_HandleTypeDef::XferCount***
  SMBUS transfer counter
- ***__IO uint32_t SMBUS_HandleTypeDef::XferOptions***
  SMBUS transfer options
- ***__IO uint32_t SMBUS_HandleTypeDef::PreviousState***
  SMBUS communication Previous state
- ***HAL_LockTypeDef SMBUS_HandleTypeDef::Lock***
  SMBUS locking object
- ***__IO uint32_t SMBUS_HandleTypeDef::State***
  SMBUS communication state
- ***__IO uint32_t SMBUS_HandleTypeDef::ErrorCode***
  SMBUS Error code

61.2 SMBUS Firmware driver API description

61.2.1 How to use this driver

The SMBUS HAL driver can be used as follows:

1. Declare a SMBUS_HandleTypeDef handle structure, for example: SMBUS_HandleTypeDef hsmbus;
2. Initialize the SMBUS low level resources by implementing the HAL_SMBUS_MspInit() API:
   a. Enable the SMBUSx interface clock
   b. SMBUS pins configuration
      – Enable the clock for the SMBUS GPIOs
      – Configure SMBUS pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      – Configure the SMBUSx interrupt priority
      – Enable the NVIC SMBUS IRQ Channel
3. Configure the Communication Clock Timing, Bus Timeout, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call, Nostretch mode, Peripheral mode and Packet Error Check mode in the hsmbus Init structure.
4. Initialize the SMBUS registers by calling the HAL_SMBUS_Init() API:
   – These APIs configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMBUS_MspInit(&hsmbus) API.
5. To check if target device is ready for communication, use the function HAL_SMBUS_IsDeviceReady()
6. For SMBUS IO operations, only one mode of operations is available within this driver

Interrupt mode IO operation

- Transmit in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Transmit_IT()
  – At transmission end of transfer HAL_SMBUS_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_MasterTxCpltCallback()
- Receive in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Receive_IT()
  – At reception end of transfer HAL_SMBUS_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_MasterRxCpltCallback()
- Abort a master/host SMBUS process communication with Interrupt using HAL_SMBUS_Master_Abort_IT()
  – The associated previous transfer callback is called at the end of abort process
  – mean HAL_SMBUS_MasterTxCpltCallback() in case of previous state was master transmit
  – mean HAL_SMBUS_MasterRxCpltCallback() in case of previous state was master receive
- Enable/disable the Address listen mode in slave/device or host/slave SMBUS mode using HAL_SMBUS_EnableListen_IT() HAL_SMBUS_DisableListen_IT()
  – When address slave/device SMBUS match, HAL_SMBUS_AddrCallback() is executed and user can add his own code to check the Address Match Code and the transmission direction request by master/host (Write/Read).
  – At Listen mode end HAL_SMBUS_ListenCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_ListenCpltCallback()
- Transmit in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Transmit_IT()
  – At transmission end of transfer HAL_SMBUS_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_SlaveTxCpltCallback()

- Receive in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Receive_IT()
  - At reception end of transfer HAL_SMBUS_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_SlaveRxCpltCallback()
- Enable/Disable the SMBUS alert mode using HAL_SMBUS_EnableAlert_IT() HAL_SMBUS_DisableAlert_IT()
  - When SMBUS Alert is generated HAL_SMBUS_ErrorCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Alert Error Code using function HAL_SMBUS_GetError()
- Get HAL state machine or error values using HAL_SMBUS_GetState() or HAL_SMBUS_GetError()
- In case of transfer Error, HAL_SMBUS_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Error Code using function HAL_SMBUS_GetError()

SMBUS HAL driver macros list

Below the list of most used macros in SMBUS HAL driver.

- __HAL_SMBUS_ENABLE: Enable the SMBUS peripheral
- __HAL_SMBUS_DISABLE: Disable the SMBUS peripheral
- __HAL_SMBUS_GET_FLAG: Check whether the specified SMBUS flag is set or not
- __HAL_SMBUS_CLEAR_FLAG: Clear the specified SMBUS pending flag
- __HAL_SMBUS_ENABLE_IT: Enable the specified SMBUS interrupt
- __HAL_SMBUS_DISABLE_IT: Disable the specified SMBUS interrupt

You can refer to the SMBUS HAL driver header file for more useful macros

61.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the SMBUSx peripheral:

- User must Implement HAL_SMBUS_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC).
- Call the function HAL_SMBUS_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Bus Timeout
  - Analog Filer mode
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
  - Packet Error Check mode
  - Peripheral mode

- Call the function HAL_SMBUS_DeInit() to restore the default configuration of the selected SMBUSx peripheral.
- Enable/Disable Analog/Digital filters with HAL_SMBUS_ConfigAnalogFilter() and HAL_SMBUS_ConfigDigitalFilter().

This section contains the following APIs:

- ***HAL_SMBUS_Init()***
- ***HAL_SMBUS_DeInit()***
- ***HAL_SMBUS_MspInit()***
- ***HAL_SMBUS_MspDeInit()***
- ***HAL_SMBUS_ConfigAnalogFilter()***
- ***HAL_SMBUS_ConfigDigitalFilter()***

61.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SMBUS data transfers.

1. Blocking mode function to check if device is ready for usage is:
   – HAL_SMBUS_IsDeviceReady()
2. There is only one mode of transfer:
   – Non-Blocking mode: The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. Non-Blocking mode functions with Interrupt are:
   – HAL_SMBUS_Master_Transmit_IT()
   – HAL_SMBUS_Master_Receive_IT()
   – HAL_SMBUS_Slave_Transmit_IT()
   – HAL_SMBUS_Slave_Receive_IT()
   – HAL_SMBUS_EnableListen_IT() or alias HAL_SMBUS_EnableListen_IT()
   – HAL_SMBUS_DisableListen_IT()
   – HAL_SMBUS_EnableAlert_IT()
   – HAL_SMBUS_DisableAlert_IT()
4. A set of Transfer Complete Callbacks are provided in non-Blocking mode:
   – HAL_SMBUS_MasterTxCpltCallback()
   – HAL_SMBUS_MasterRxCpltCallback()
   – HAL_SMBUS_SlaveTxCpltCallback()
   – HAL_SMBUS_SlaveRxCpltCallback()
   – HAL_SMBUS_AddrCallback()
   – HAL_SMBUS_ListenCpltCallback()
   – HAL_SMBUS_ErrorCallback()

This section contains the following APIs:

- ***HAL_SMBUS_Master_Transmit_IT()***
- ***HAL_SMBUS_Master_Receive_IT()***
- ***HAL_SMBUS_Master_Abort_IT()***
- ***HAL_SMBUS_Slave_Transmit_IT()***
- ***HAL_SMBUS_Slave_Receive_IT()***
- ***HAL_SMBUS_EnableListen_IT()***
- ***HAL_SMBUS_DisableListen_IT()***
- ***HAL_SMBUS_EnableAlert_IT()***
- ***HAL_SMBUS_DisableAlert_IT()***
- ***HAL_SMBUS_IsDeviceReady()***

61.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_SMBUS_GetState()***
- ***HAL_SMBUS_GetError()***

61.2.5 Detailed description of functions

HAL_SMBUS_Init

Function name	**HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)**
Function description	Initialize the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and initialize the associated handle.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **HAL:** status

HAL_SMBUS_DeInit

Function name	**HAL_StatusTypeDef HAL_SMBUS_DeInit (SMBUS_HandleTypeDef * hsmbus)**
Function description	DeInitialize the SMBUS peripheral.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **HAL:** status

HAL_SMBUS_MspInit

Function name	**void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)**
Function description	Initialize the SMBUS MSP.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None:**

HAL_SMBUS_MspDeInit

Function name	**void HAL_SMBUS_MspDeInit (SMBUS_HandleTypeDef * hsmbus)**
Function description	DeInitialize the SMBUS MSP.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values	• **None:**

HAL_SMBUS_ConfigAnalogFilter

Function name	**HAL_StatusTypeDef HAL_SMBUS_ConfigAnalogFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t AnalogFilter)**
Function description	Configure Analog noise filter.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **AnalogFilter:** This parameter can be one of the following values: – SMBUS_ANALOGFILTER_ENABLE – SMBUS_ANALOGFILTER_DISABLE
Return values	• **HAL:** status

HAL_SMBUS_ConfigDigitalFilter

Function name	**HAL_StatusTypeDef HAL_SMBUS_ConfigDigitalFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t DigitalFilter)**
Function description	Configure Digital noise filter.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.
Return values	• **HAL:** status

HAL_SMBUS_IsDeviceReady

Function name	**HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)**
Function description	Check if target device is ready for communication.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **Trials:** Number of trials • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_SMBUS_Master_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**

Function description	Transmit in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition
Return values	• **HAL:** status

HAL_SMBUS_Master_Receive_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**
Function description	Receive in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition
Return values	• **HAL:** status

HAL_SMBUS_Master_Abort_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)**
Function description	Abort a master/host SMBUS process communication with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface
Return values	• **HAL:** status
Notes	• This abort can be called only if state is ready

HAL_SMBUS_Slave_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**
Function description	Transmit in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition
Return values	• **HAL:** status

HAL_SMBUS_Slave_Receive_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**
Function description	Receive in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition
Return values	• **HAL:** status

HAL_SMBUS_EnableAlert_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)**
Function description	Enable the SMBUS alert mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	• **HAL:** status

HAL_SMBUS_DisableAlert_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)**
Function description	Disable the SMBUS alert mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified

SMBUSx peripheral.

Return values	• **HAL:** status

HAL_SMBUS_EnableListen_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)**
Function description	Enable the Address listen mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **HAL:** status

HAL_SMBUS_DisableListen_IT

Function name	**HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)**
Function description	Disable the Address listen mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **HAL:** status

HAL_SMBUS_EV_IRQHandler

Function name	**void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)**
Function description	Handle SMBUS event interrupt request.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None:**

HAL_SMBUS_ER_IRQHandler

Function name	**void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)**
Function description	Handle SMBUS error interrupt request.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None:**

HAL_SMBUS_MasterTxCpltCallback

Function name	**void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**

Function description	Master Tx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None:**

HAL_SMBUS_MasterRxCpltCallback

Function name	**void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	Master Rx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None:**

HAL_SMBUS_SlaveTxCpltCallback

Function name	**void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	Slave Tx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None:**

HAL_SMBUS_SlaveRxCpltCallback

Function name	**void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	Slave Rx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None:**

HAL_SMBUS_AddrCallback

Function name	**void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)**
Function description	Slave Address Match callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **TransferDirection:** Master request Transfer Direction (Write/Read) • **AddrMatchCode:** Address Match Code

Return values	• **None:**

HAL_SMBUS_ListenCpltCallback

Function name	**void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	Listen Complete callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None:**

HAL_SMBUS_ErrorCallback

Function name	**void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)**
Function description	SMBUS error callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **None:**

HAL_SMBUS_GetState

Function name	**uint32_t HAL_SMBUS_GetState (SMBUS_HandleTypeDef * hsmbus)**
Function description	Return the SMBUS handle state.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **HAL:** state

HAL_SMBUS_GetError

Function name	**uint32_t HAL_SMBUS_GetError (SMBUS_HandleTypeDef * hsmbus)**
Function description	Return the SMBUS error code.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• **SMBUS:** Error Code

61.3 SMBUS Firmware driver defines

61.3.1 SMBUS

SMBUS addressing mode

SMBUS_ADDRESSINGMODE_7BIT

SMBUS_ADDRESSINGMODE_10BIT

SMBUS Analog Filter

SMBUS_ANALOGFILTER_ENABLE

SMBUS_ANALOGFILTER_DISABLE

SMBUS dual addressing mode

SMBUS_DUALADDRESS_DISABLE

SMBUS_DUALADDRESS_ENABLE

SMBUS Error Code definition

HAL_SMBUS_ERROR_NONE	No error
HAL_SMBUS_ERROR_BERR	BERR error
HAL_SMBUS_ERROR_ARLO	ARLO error
HAL_SMBUS_ERROR_ACKF	ACKF error
HAL_SMBUS_ERROR_OVR	OVR error
HAL_SMBUS_ERROR_HALTIMEOUT	Timeout error
HAL_SMBUS_ERROR_BUSTIMEOUT	Bus Timeout error
HAL_SMBUS_ERROR_ALERT	Alert error
HAL_SMBUS_ERROR_PECERR	PEC error

SMBUS Exported Macros

__HAL_SMBUS_RESET_HANDLE_STATE

Description:

- Reset SMBUS handle state.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_ENABLE_IT

Description:

- Enable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:

- SMBUS_IT_ERRI Errors interrupt enable
- SMBUS_IT_TCI Transfer complete interrupt enable
- SMBUS_IT_STOPI STOP detection interrupt enable
- SMBUS_IT_NACKI NACK received interrupt enable
- SMBUS_IT_ADDRI Address match interrupt enable
- SMBUS_IT_RXI RX interrupt enable
- SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_DISABLE_IT

Description:

- Disable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_GET_IT_SOURCE

Description:

- Check whether the specified SMBUS interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

- __INTERRUPT__: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- The: new state of __IT__ (TRUE or FALSE).

SMBUS_FLAG_MASK

Description:

- Check whether the specified SMBUS flag is set or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMBUS_FLAG_TXE Transmit data register empty
  - SMBUS_FLAG_TXIS Transmit interrupt status
  - SMBUS_FLAG_RXNE Receive data register not empty
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_TC Transfer complete (master mode)
  - SMBUS_FLAG_TCR Transfer complete reload
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun

- SMBUS_FLAG_PECERR PEC error in reception
- SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
- SMBUS_FLAG_ALERT SMBus alert
- SMBUS_FLAG_BUSY Bus busy
- SMBUS_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SMBUS_GET_FLAG

__HAL_SMBUS_CLEAR_FLAG

Description:

- Clear the SMBUS pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_SMBUS_ENABLE

Description:

- Enable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_DISABLE

Description:

- Disable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_GENERATE_NACK

Description:

- Generate a Non-Acknowledge SMBUS peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

SMBUS Flag definition

SMBUS_FLAG_TXE

SMBUS_FLAG_TXIS

SMBUS_FLAG_RXNE

SMBUS_FLAG_ADDR

SMBUS_FLAG_AF

SMBUS_FLAG_STOPF

SMBUS_FLAG_TC

SMBUS_FLAG_TCR

SMBUS_FLAG_BERR

SMBUS_FLAG_ARLO

SMBUS_FLAG_OVR

SMBUS_FLAG_PECERR

SMBUS_FLAG_TIMEOUT

SMBUS_FLAG_ALERT

SMBUS_FLAG_BUSY

SMBUS_FLAG_DIR

SMBUS general call addressing mode

SMBUS_GENERALCALL_DISABLE

SMBUS_GENERALCALL_ENABLE

SMBUS Interrupt configuration definition

SMBUS_IT_ERRI

SMBUS_IT_TCI

SMBUS_IT_STOPI

SMBUS_IT_NACKI

SMBUS_IT_ADDRI

SMBUS_IT_RXI

SMBUS_IT_TXI

SMBUS_IT_TX

SMBUS_IT_RX

SMBUS_IT_ALERT

SMBUS_IT_ADDR

SMBUS nostretch mode

SMBUS_NOSTRETCH_DISABLE

SMBUS_NOSTRETCH_ENABLE

SMBUS ownaddress2 masks

SMBUS_OA2_NOMASK

SMBUS_OA2_MASK01

SMBUS_OA2_MASK02

SMBUS_OA2_MASK03

SMBUS_OA2_MASK04

SMBUS_OA2_MASK05

SMBUS_OA2_MASK06

SMBUS_OA2_MASK07

SMBUS packet error check mode

SMBUS_PEC_DISABLE

SMBUS_PEC_ENABLE

SMBUS peripheral mode

SMBUS_PERIPHERAL_MODE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP

SMBUS ReloadEndMode definition

SMBUS_SOFTEND_MODE

SMBUS_RELOAD_MODE

SMBUS_AUTOEND_MODE

SMBUS_SENDPEC_MODE

SMBUS StartStopMode definition

SMBUS_NO_STARTSTOP

SMBUS_GENERATE_STOP

SMBUS_GENERATE_START_READ

SMBUS_GENERATE_START_WRITE

SMBUS XferOptions definition

SMBUS_FIRST_FRAME

SMBUS_NEXT_FRAME

SMBUS_FIRST_AND_LAST_FRAME_NO_PEC

SMBUS_LAST_FRAME_NO_PEC

SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC

SMBUS_LAST_FRAME_WITH_PEC

SMBUS_OTHER_FRAME_NO_PEC

SMBUS_OTHER_FRAME_WITH_PEC

SMBUS_OTHER_AND_LAST_FRAME_NO_PEC

SMBUS_OTHER_AND_LAST_FRAME_WITH_PEC

62 HAL SPI Generic Driver

62.1 SPI Firmware driver registers structures

62.1.1 SPI_InitTypeDef

Data Fields

- ***uint32_t Mode***
- ***uint32_t Direction***
- ***uint32_t DataSize***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRatePrescaler***
- ***uint32_t FirstBit***
- ***uint32_t TIMode***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t NSSPMode***

Field Documentation

- ***uint32_t SPI_InitTypeDef::Mode***
  Specifies the SPI operating mode. This parameter can be a value of ***SPI_Mode***
- ***uint32_t SPI_InitTypeDef::Direction***
  Specifies the SPI bidirectional mode state. This parameter can be a value of ***SPI_Direction***
- ***uint32_t SPI_InitTypeDef::DataSize***
  Specifies the SPI data size. This parameter can be a value of ***SPI_Data_Size***
- ***uint32_t SPI_InitTypeDef::CLKPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_Clock_Polarity***
- ***uint32_t SPI_InitTypeDef::CLKPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_Clock_Phase***
- ***uint32_t SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_Slave_Select_management***
- ***uint32_t SPI_InitTypeDef::BaudRatePrescaler***
  Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_BaudRate_Prescaler***
  Note: The communication clock is derived from the master clock. The slave clock does not need to be set.
- ***uint32_t SPI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_MSB_LSB_transmission***
- ***uint32_t SPI_InitTypeDef::TIMode***
  Specifies if the TI mode is enabled or not. This parameter can be a value of ***SPI_TI_mode***

- ***uint32_t SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_CRC_Calculation***
- ***uint32_t SPI_InitTypeDef::CRCPolynomial***
  Specifies the polynomial used for the CRC calculation. This parameter must be an odd number between Min_Data = 1 and Max_Data = 65535
- ***uint32_t SPI_InitTypeDef::CRCLength***
  Specifies the CRC Length used for the CRC calculation. CRC Length is only used with Data8 and Data16, not other data size This parameter can be a value of ***SPI_CRC_length***
- ***uint32_t SPI_InitTypeDef::NSSPMode***
  Specifies whether the NSSP signal is enabled or not . This parameter can be a value of ***SPI_NSSP_Mode*** This mode is activated by the NSSP bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx_CR1 CPHA = 0, CPOL setting is ignored)..

62.1.2 __SPI_HandleTypeDef

Data Fields

- ***SPI_TypeDef * Instance***
- ***SPI_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint32_t CRCSize***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SPI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***SPI_TypeDef* __SPI_HandleTypeDef::Instance***
  SPI registers base address
- ***SPI_InitTypeDef __SPI_HandleTypeDef::Init***
  SPI communication parameters
- ***uint8_t* __SPI_HandleTypeDef::pTxBuffPtr***
  Pointer to SPI Tx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::TxXferSize***
  SPI Tx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::TxXferCount***
  SPI Tx Transfer Counter
- ***uint8_t* __SPI_HandleTypeDef::pRxBuffPtr***
  Pointer to SPI Rx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::RxXferSize***
  SPI Rx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::RxXferCount***
  SPI Rx Transfer Counter

- ***uint32_t __SPI_HandleTypeDef::CRCSize***
  SPI CRC size used for the transfer
- ***void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Rx ISR
- ***void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Tx ISR
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx***
  SPI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx***
  SPI Rx DMA Handle parameters
- ***HAL_LockTypeDef __SPI_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State***
  SPI communication state
- ***__IO uint32_t __SPI_HandleTypeDef::ErrorCode***
  SPI Error code

62.2 SPI Firmware driver API description

62.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit() API:
   a. Enable the SPIx interface clock
   b. SPI pins configuration
      – Enable the clock for the SPI GPIOs
      – Configure these SPI pins as alternate function push-pull
   c. NVIC configuration if you need to use interrupt process
      – Configure the SPIx interrupt priority
      – Enable the NVIC SPI IRQ handle
   d. DMA Configuration if you need to use DMA process
      – Declare a DMA_HandleTypeDef handle structure for the transmit or receive Stream/Channel
      – Enable the DMAx clock
      – Configure the DMA handle parameters
      – Configure the DMA Tx or Rx Stream/Channel
      – Associate the initialized hdma_tx handle to the hspi DMA Tx or Rx handle
      – Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Stream/Channel
3. Program the Mode, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL_SPI_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit() API.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
   a. Master 2Lines RxOnly
   b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled

3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL_SPI_DMAPause()/ HAL_SPI_DMAStop() only under the SPI callbacks

Master Receive mode restriction:

1. In Master unidirectional receive-only mode (MSTR =1, BIDIMODE=0, RXONLY=0) or bidirectional receive mode (MSTR=1, BIDIMODE=1, BIDIOE=0), to ensure that the SPI does not initiate a new transfer the following procedure has to be respected:
   a. HAL_SPI_DeInit()
   b. HAL_SPI_Init()

The HAL drivers do not allow reaching all supported SPI frequencies in the different SPI modes. Refer to the source code (stm32xxxx_hal_spi.c header) to get a summary of the maximum SPI frequency that can be reached with a data size of 8 or 16 bits, depending on the APBx peripheral clock frequency (fPCLK) used by the SPI instance.

62.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPIx peripheral:

- User must implement HAL_SPI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPI_Init() to configure the selected device with the selected configuration:
  - Mode
  - Direction
  - Data Size
  - Clock Polarity and Phase
  - NSS Management
  - BaudRate Prescaler
  - FirstBit
  - TIMode
  - CRC Calculation
  - CRC Polynomial if CRC enabled
  - CRC Length, used only with Data8 and Data16
  - FIFO reception threshold
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

- ***HAL_SPI_Init()***
- ***HAL_SPI_DeInit()***
- ***HAL_SPI_MspInit()***
- ***HAL_SPI_MspDeInit()***

62.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SPI data transfers.

The SPI supports master and slave mode:

1. There are two modes of transfer:
   - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be

indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_SPI_ErrorCallback()user callback will be executed when a communication error is detected

2. APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.

This section contains the following APIs:

- ***HAL_SPI_Transmit()***
- ***HAL_SPI_Receive()***
- ***HAL_SPI_TransmitReceive()***
- ***HAL_SPI_Transmit_IT()***
- ***HAL_SPI_Receive_IT()***
- ***HAL_SPI_TransmitReceive_IT()***
- ***HAL_SPI_Transmit_DMA()***
- ***HAL_SPI_Receive_DMA()***
- ***HAL_SPI_TransmitReceive_DMA()***
- ***HAL_SPI_Abort()***
- ***HAL_SPI_Abort_IT()***
- ***HAL_SPI_DMAPause()***
- ***HAL_SPI_DMAResume()***
- ***HAL_SPI_DMAStop()***
- ***HAL_SPI_IRQHandler()***
- ***HAL_SPI_TxCpltCallback()***
- ***HAL_SPI_RxCpltCallback()***
- ***HAL_SPI_TxRxCpltCallback()***
- ***HAL_SPI_TxHalfCpltCallback()***
- ***HAL_SPI_RxHalfCpltCallback()***
- ***HAL_SPI_TxRxHalfCpltCallback()***
- ***HAL_SPI_ErrorCallback()***
- ***HAL_SPI_AbortCpltCallback()***

62.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to control the SPI.

- HAL_SPI_GetState() API can be helpful to check in run-time the state of the SPI peripheral
- HAL_SPI_GetError() check in run-time Errors occurring during communication

This section contains the following APIs:

- ***HAL_SPI_GetState()***
- ***HAL_SPI_GetError()***

62.2.5 Detailed description of functions

HAL_SPI_Init

Function name	**HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)**
Function description	Initialize the SPI according to the specified parameters in the SPI_InitTypeDef and initialize the associated handle.

Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **HAL:** status

HAL_SPI_DeInit

Function name	**HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)**
Function description	De-Initialize the SPI peripheral.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **HAL:** status

HAL_SPI_MspInit

Function name	**void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)**
Function description	Initialize the SPI MSP.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_MspDeInit

Function name	**void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)**
Function description	De-Initialize the SPI MSP.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_Transmit

Function name	**HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Transmit an amount of data in blocking mode.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent • **Timeout:** Timeout duration
Return values	• **HAL:** status

HAL_SPI_Receive

Function name	**HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be received
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_SPI_TransmitReceive

Function name
HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description
Transmit and Receive an amount of data in blocking mode.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** amount of data to be sent and received
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_SPI_Transmit_IT

Function name
HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description
Transmit an amount of data in non-blocking mode with Interrupt.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values
- **HAL:** status

HAL_SPI_Receive_IT

Function name
HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description
Receive an amount of data in non-blocking mode with Interrupt.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values
- **HAL:** status

HAL_SPI_TransmitReceive_IT

Function name
HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description	Transmit and Receive an amount of data in non-blocking mode with Interrupt.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pTxData:** pointer to transmission data buffer • **pRxData:** pointer to reception data buffer • **Size:** amount of data to be sent and received
Return values	• **HAL:** status

HAL_SPI_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**
Function description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• **HAL:** status

HAL_SPI_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in non-blocking mode with DMA.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• **HAL:** status
Notes	• In case of MASTER mode and SPI_DIRECTION_2LINES direction, hdmatx shall be defined. • When the CRC feature is enabled the pData Length must be Size + 1.

HAL_SPI_TransmitReceive_DMA

Function name	**HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**
Function description	Transmit and Receive an amount of data in non-blocking mode with DMA.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pTxData:** pointer to transmission data buffer • **pRxData:** pointer to reception data buffer • **Size:** amount of data to be sent

Return values
- **HAL:** status

Notes
- When the CRC feature is enabled the pRxData Length must be Size + 1

HAL_SPI_DMAPause

Function name: **HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)**

Function description: Pause the DMA Transfer.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values
- **HAL:** status

HAL_SPI_DMAResume

Function name: **HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)**

Function description: Resume the DMA Transfer.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values
- **HAL:** status

HAL_SPI_DMAStop

Function name: **HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)**

Function description: Stop the DMA Transfer.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values
- **HAL:** status

HAL_SPI_Abort

Function name: **HAL_StatusTypeDef HAL_SPI_Abort (SPI_HandleTypeDef * hspi)**

Function description: Abort ongoing transfer (blocking mode).

Parameters
- **hspi:** SPI handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode. This procedure performs following operations: Disable SPI Interrupts (depending of transfer direction)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode: when exiting

function, Abort is considered as completed.

HAL_SPI_Abort_IT

Function name	**HAL_StatusTypeDef HAL_SPI_Abort_IT (SPI_HandleTypeDef * hspi)**
Function description	Abort ongoing transfer (Interrupt mode).
Parameters	• **hspi:** SPI handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode. This procedure performs following operations: Disable SPI Interrupts (depending of transfer direction)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SPI_IRQHandler

Function name	**void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)**
Function description	Handle SPI interrupt request.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	• **None:**

HAL_SPI_TxCpltCallback

Function name	**void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx Transfer completed callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_RxCpltCallback

Function name	**void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Rx Transfer completed callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_TxRxCpltCallback

Function name	**void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx and Rx Transfer completed callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_TxHalfCpltCallback

Function name	**void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx Half Transfer completed callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_RxHalfCpltCallback

Function name	**void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Rx Half Transfer completed callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_TxRxHalfCpltCallback

Function name	**void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx and Rx Half Transfer callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_ErrorCallback

Function name	**void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)**
Function description	SPI error callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_AbortCpltCallback

Function name	**void HAL_SPI_AbortCpltCallback (SPI_HandleTypeDef * hspi)**

Function description	SPI Abort Complete callback.
Parameters	• **hspi:** SPI handle.
Return values	• **None:**

HAL_SPI_GetState

Function name	**HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)**
Function description	Return the SPI handle state.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **SPI:** state

HAL_SPI_GetError

Function name	**uint32_t HAL_SPI_GetError (SPI_HandleTypeDef * hspi)**
Function description	Return the SPI error code.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **SPI:** error code in bitmap format

62.3 SPI Firmware driver defines

62.3.1 SPI

SPI BaudRate Prescaler

SPI_BAUDRATEPRESCALER_2

SPI_BAUDRATEPRESCALER_4

SPI_BAUDRATEPRESCALER_8

SPI_BAUDRATEPRESCALER_16

SPI_BAUDRATEPRESCALER_32

SPI_BAUDRATEPRESCALER_64

SPI_BAUDRATEPRESCALER_128

SPI_BAUDRATEPRESCALER_256

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

SPI CRC Length

SPI_CRC_LENGTH_DATASIZE

SPI_CRC_LENGTH_8BIT

SPI_CRC_LENGTH_16BIT

SPI Data Size

SPI_DATASIZE_4BIT

SPI_DATASIZE_5BIT

SPI_DATASIZE_6BIT

SPI_DATASIZE_7BIT

SPI_DATASIZE_8BIT

SPI_DATASIZE_9BIT

SPI_DATASIZE_10BIT

SPI_DATASIZE_11BIT

SPI_DATASIZE_12BIT

SPI_DATASIZE_13BIT

SPI_DATASIZE_14BIT

SPI_DATASIZE_15BIT

SPI_DATASIZE_16BIT

SPI Direction Mode

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

SPI Error Code

HAL_SPI_ERROR_NONE	No error
HAL_SPI_ERROR_MODF	MODF error
HAL_SPI_ERROR_CRC	CRC error
HAL_SPI_ERROR_OVR	OVR error
HAL_SPI_ERROR_FRE	FRE error
HAL_SPI_ERROR_DMA	DMA transfer error
HAL_SPI_ERROR_FLAG	Error on RXNE/TXE/BSY/FTLVL/FRLVL Flag
HAL_SPI_ERROR_ABORT	Error during SPI Abort procedure

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE

Description:

- Reset SPI handle state.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT

Description:

- Enable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  – SPI_IT_TXE: Tx buffer empty interrupt enable
  – SPI_IT_RXNE: RX buffer not empty interrupt enable
  – SPI_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPIx where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  – SPI_IT_TXE: Tx buffer empty interrupt enable
  – SPI_IT_RXNE: RX buffer not empty interrupt enable
  – SPI_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_SPI_GET_IT_SOURCE

Description:

- Check whether the specified SPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

- __INTERRUPT__: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_SPI_GET_FLAG

Description:

- Check whether the specified SPI flag is set or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPI_FLAG_RXNE: Receive buffer not empty flag
  - SPI_FLAG_TXE: Transmit buffer empty flag
  - SPI_FLAG_CRCERR: CRC error flag
  - SPI_FLAG_MODF: Mode fault flag
  - SPI_FLAG_OVR: Overrun flag
  - SPI_FLAG_BSY: Busy flag
  - SPI_FLAG_FRE: Frame format error flag
  - SPI_FLAG_FTLVL: SPI fifo transmission level
  - SPI_FLAG_FRLVL: SPI fifo reception level

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SPI_CLEAR_CRCERRFLAG

Description:

- Clear the SPI CRCERR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_MODFFLAG

Description:

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_OVRFLAG

Description:

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_FREFLAG

Description:

- Clear the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE

Description:

- Enable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_DISABLE

Description:

- Disable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

SPI FIFO Reception Threshold

SPI_RXFIFO_THRESHOLD

SPI_RXFIFO_THRESHOLD_QF

SPI_RXFIFO_THRESHOLD_HF

SPI Flags Definition

SPI_FLAG_RXNE

SPI_FLAG_TXE

SPI_FLAG_BSY

SPI_FLAG_CRCERR

SPI_FLAG_MODF

SPI_FLAG_OVR

SPI_FLAG_FRE

SPI_FLAG_FTLVL

SPI_FLAG_FRLVL

SPI Interrupt Definition

SPI_IT_TXE

SPI_IT_RXNE

SPI_IT_ERR

SPI Mode

SPI_MODE_SLAVE

SPI_MODE_MASTER

SPI MSB LSB Transmission

SPI_FIRSTBIT_MSB

SPI_FIRSTBIT_LSB

SPI NSS Pulse Mode

SPI_NSS_PULSE_ENABLE

SPI_NSS_PULSE_DISABLE

SPI Reception FIFO Status Level

SPI_FRLVL_EMPTY

SPI_FRLVL_QUARTER_FULL

SPI_FRLVL_HALF_FULL

SPI_FRLVL_FULL

SPI Slave Select Management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

SPI TI Mode

SPI_TIMODE_DISABLE

SPI_TIMODE_ENABLE

SPI Transmission FIFO Status Level

SPI_FTLVL_EMPTY

SPI_FTLVL_QUARTER_FULL

SPI_FTLVL_HALF_FULL

SPI_FTLVL_FULL

63 HAL SPI Extension Driver

63.1 SPIEx Firmware driver API description

63.1.1 IO operation functions

This subsection provides a set of extended functions to manage the SPI data transfers.

1. Rx data flush function:
   - HAL_SPIEx_FlushRxFifo()

This section contains the following APIs:

- ***HAL_SPIEx_FlushRxFifo()***

63.1.2 Detailed description of functions

HAL_SPIEx_FlushRxFifo

Function name	**HAL_StatusTypeDef HAL_SPIEx_FlushRxFifo (SPI_HandleTypeDef * hspi)**
Function description	Flush the RX fifo.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	• **HAL:** status

64 HAL SRAM Generic Driver

64.1 SRAM Firmware driver registers structures

64.1.1 SRAM_HandleTypeDef

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SRAM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdma***

Field Documentation

- ***FMC_NORSRAM_TypeDef* SRAM_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* SRAM_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef SRAM_HandleTypeDef::Init***
  SRAM device control configuration parameters
- ***HAL_LockTypeDef SRAM_HandleTypeDef::Lock***
  SRAM locking object
- ***__IO HAL_SRAM_StateTypeDef SRAM_HandleTypeDef::State***
  SRAM device access state
- ***DMA_HandleTypeDef* SRAM_HandleTypeDef::hdma***
  Pointer DMA handler

64.2 SRAM Firmware driver API description

64.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM_HandleTypeDef handle structure, for example: SRAM_HandleTypeDef hsram; and:
   – Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
   – Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two FMC_NORSRAM_TimingTypeDef structures, for both normal and extended mode timings; for example: FMC_NORSRAM_TimingTypeDef Timing and FMC_NORSRAM_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function HAL_SRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SRAM_MspInit()

b. Control register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Init()
c. Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Timing_Init()
d. Extended mode Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Extended_Timing_Init()
e. Enable the SRAM device using the macro __FMC_NORSRAM_ENABLE()

4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   - HAL_SRAM_Read()/HAL_SRAM_Write() for polling read/write access
   - HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs HAL_SRAM_WriteOperation_Enable()/ HAL_SRAM_WriteOperation_Disable() to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function HAL_SRAM_GetState()

64.2.2 SRAM Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory.

This section contains the following APIs:

- ***HAL_SRAM_Init()***
- ***HAL_SRAM_DeInit()***
- ***HAL_SRAM_MspInit()***
- ***HAL_SRAM_MspDeInit()***
- ***HAL_SRAM_DMA_XferCpltCallback()***
- ***HAL_SRAM_DMA_XferErrorCallback()***

64.2.3 SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

This section contains the following APIs:

- ***HAL_SRAM_Read_8b()***
- ***HAL_SRAM_Write_8b()***
- ***HAL_SRAM_Read_16b()***
- ***HAL_SRAM_Write_16b()***
- ***HAL_SRAM_Read_32b()***
- ***HAL_SRAM_Write_32b()***
- ***HAL_SRAM_Read_DMA()***
- ***HAL_SRAM_Write_DMA()***

64.2.4 SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.

This section contains the following APIs:

- ***HAL_SRAM_WriteOperation_Enable()***
- ***HAL_SRAM_WriteOperation_Disable()***

64.2.5 SRAM State functions

This subsection permits to get in run-time the status of the SRAM controller and the data flow.

This section contains the following APIs:

- ***HAL_SRAM_GetState()***

64.2.6 Detailed description of functions

HAL_SRAM_Init

Function name	**HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)**
Function description	Perform the SRAM device initialization sequence.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **Timing:** Pointer to SRAM control timing structure • **ExtTiming:** Pointer to SRAM extended mode timing structure
Return values	• **HAL:** status

HAL_SRAM_DeInit

Function name	**HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)**
Function description	Perform the SRAM device de-initialization sequence.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **HAL:** status

HAL_SRAM_MspInit

Function name	**void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)**
Function description	Initialize the SRAM MSP.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **None:**

HAL_SRAM_MspDeInit

Function name	**void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)**
Function description	DeInitialize the SRAM MSP.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **None:**

HAL_SRAM_DMA_XferCpltCallback

Function name	**void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)**
Function description	DMA transfer complete callback.
Parameters	• **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **None:**

HAL_SRAM_DMA_XferErrorCallback

Function name	**void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)**
Function description	DMA transfer complete error callback.
Parameters	• **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **None:**

HAL_SRAM_Read_8b

Function name	**HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)**
Function description	Read 8-bit buffer from SRAM memory.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **pAddress:** Pointer to read start address • **pDstBuffer:** Pointer to destination buffer • **BufferSize:** Size of the buffer to read from memory
Return values	• **HAL:** status

HAL_SRAM_Write_8b

Function name	**HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)**
Function description	Write 8-bit buffer to SRAM memory.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **pAddress:** Pointer to write start address • **pSrcBuffer:** Pointer to source buffer to write • **BufferSize:** Size of the buffer to write to memory
Return values	• **HAL:** status

HAL_SRAM_Read_16b

Function name	**HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t**

	*** pDstBuffer, uint32_t BufferSize)**
Function description	Read 16-bit buffer from SRAM memory.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **pAddress:** Pointer to read start address • **pDstBuffer:** Pointer to destination buffer • **BufferSize:** Size of the buffer to read from memory
Return values	• **HAL:** status

HAL_SRAM_Write_16b

Function name	**HAL_StatusTypeDef HAL_SRAM_Write_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)**
Function description	Write 16-bit buffer to SRAM memory.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **pAddress:** Pointer to write start address • **pSrcBuffer:** Pointer to source buffer to write • **BufferSize:** Size of the buffer to write to memory
Return values	• **HAL:** status

HAL_SRAM_Read_32b

Function name	**HAL_StatusTypeDef HAL_SRAM_Read_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)**
Function description	Read 32-bit buffer from SRAM memory.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **pAddress:** Pointer to read start address • **pDstBuffer:** Pointer to destination buffer • **BufferSize:** Size of the buffer to read from memory
Return values	• **HAL:** status

HAL_SRAM_Write_32b

Function name	**HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)**
Function description	Write 32-bit buffer to SRAM memory.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **pAddress:** Pointer to write start address • **pSrcBuffer:** Pointer to source buffer to write • **BufferSize:** Size of the buffer to write to memory
Return values	• **HAL:** status

HAL_SRAM_Read_DMA

Function name	**HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)**
Function description	Read a Word data buffer from the SRAM memory using DMA transfer.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **pAddress:** Pointer to read start address • **pDstBuffer:** Pointer to destination buffer • **BufferSize:** Size of the buffer to read from memory
Return values	• **HAL:** status

HAL_SRAM_Write_DMA

Function name	**HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)**
Function description	Write a Word data buffer to SRAM memory using DMA transfer.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **pAddress:** Pointer to write start address • **pSrcBuffer:** Pointer to source buffer to write • **BufferSize:** Size of the buffer to write to memory
Return values	• **HAL:** status

HAL_SRAM_WriteOperation_Enable

Function name	**HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)**
Function description	Enable dynamically SRAM write operation.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **HAL:** status

HAL_SRAM_WriteOperation_Disable

Function name	**HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)**
Function description	Disable dynamically SRAM write operation.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **HAL:** status

HAL_SRAM_GetState

Function name	**HAL_SRAM_StateTypeDef HAL_SRAM_GetState**

	(SRAM_HandleTypeDef * hsram)
Function description	Return the SRAM controller handle state.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **HAL:** state

64.3 SRAM Firmware driver defines

64.3.1 SRAM

SRAM Exported Macros

__HAL_SRAM_RESET_HANDLE_STATE	**Description:** • Reset SRAM handle state. **Parameters:** • __HANDLE__: SRAM handle **Return value:** • None

65 HAL TIM Generic Driver

65.1 TIM Firmware driver registers structures

65.1.1 TIM_Base_InitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Period***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***
- ***uint32_t AutoReloadPreload***

Field Documentation

- ***uint32_t TIM_Base_InitTypeDef::Prescaler***
  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_Base_InitTypeDef::CounterMode***
  Specifies the counter mode. This parameter can be a value of ***TIM_Counter_Mode***
- ***uint32_t TIM_Base_InitTypeDef::Period***
  Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::ClockDivision***
  Specifies the clock division. This parameter can be a value of ***TIM_ClockDivision***
- ***uint32_t TIM_Base_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:the number of PWM periods in edge-aligned modethe number of half PWM period in center-aligned mode This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_Base_InitTypeDef::AutoReloadPreload***
  Specifies the auto-reload preload. This parameter can be a value of ***TIM_AutoReloadPreload***

65.1.2 TIM_OC_InitTypeDef

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCFastMode***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of
  TIM_Output_Compare_and_PWM_modes
- ***uint32_t TIM_OC_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of
  TIM_Output_Compare_Polarity
- ***uint32_t TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of
  TIM_Output_Compare_N_Polarity
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***
  Specifies the Fast mode state. This parameter can be a value of
  TIM_Output_Fast_State
  Note: This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32_t TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.

65.1.3 TIM_OnePulse_InitTypeDef

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of
  TIM_Output_Compare_and_PWM_modes
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OnePulse_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of
  TIM_Output_Compare_Polarity
- ***uint32_t TIM_OnePulse_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of
  TIM_Output_Compare_N_Polarity
  Note: This parameter is valid only for TIM1 and TIM8.

- ***uint32_t TIM_OnePulse_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OnePulse_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

65.1.4 TIM_IC_InitTypeDef

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_IC_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

65.1.5 TIM_Encoder_InitTypeDef

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1Selection***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2Selection***
- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t TIM_Encoder_InitTypeDef::EncoderMode***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Mode***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

65.1.6 TIM_ClockConfigTypeDef

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPrescaler***
- ***uint32_t ClockFilter***

Field Documentation

- ***uint32_t TIM_ClockConfigTypeDef::ClockSource***
  TIM clock sources This parameter can be a value of ***TIM_Clock_Source***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPolarity***
  TIM clock polarity This parameter can be a value of ***TIM_Clock_Polarity***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPrescaler***
  TIM clock prescaler This parameter can be a value of ***TIM_Clock_Prescaler***
- ***uint32_t TIM_ClockConfigTypeDef::ClockFilter***
  TIM clock filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

65.1.7 TIM_ClearInputConfigTypeDef

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***
  TIM clear Input state This parameter can be ENABLE or DISABLE
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***
  TIM clear Input sources This parameter can be a value of ***TIM_ClearInput_Source***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***
  TIM Clear Input polarity This parameter can be a value of ***TIM_ClearInput_Polarity***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***
  TIM Clear Input prescaler This parameter can be a value of ***TIM_ClearInput_Prescaler***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***
  TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

65.1.8 TIM_MasterConfigTypeDef

Data Fields

- ***uint32_t MasterOutputTrigger***
- ***uint32_t MasterOutputTrigger2***
- ***uint32_t MasterSlaveMode***

Field Documentation

- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger***
  Trigger output (TRGO) selection This parameter can be a value of ***TIM_Master_Mode_Selection***
- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger2***
  Trigger output2 (TRGO2) selection This parameter can be a value of ***TIM_Master_Mode_Selection_2***
- ***uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode***
  Master/slave mode selection This parameter can be a value of ***TIM_Master_Slave_Mode***

65.1.9 TIM_SlaveConfigTypeDef

Data Fields

- ***uint32_t SlaveMode***
- ***uint32_t InputTrigger***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerPrescaler***
- ***uint32_t TriggerFilter***

Field Documentation

- ***uint32_t TIM_SlaveConfigTypeDef::SlaveMode***
  Slave mode selection This parameter can be a value of ***TIM_Slave_Mode***
- ***uint32_t TIM_SlaveConfigTypeDef::InputTrigger***
  Input Trigger source This parameter can be a value of ***TIM_Trigger_Selection***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity***
  Input Trigger polarity This parameter can be a value of ***TIM_Trigger_Polarity***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler***
  Input trigger prescaler This parameter can be a value of ***TIM_Trigger_Prescaler***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerFilter***
  Input trigger filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

65.1.10 TIM_BreakDeadTimeConfigTypeDef

Data Fields

- ***uint32_t OffStateRunMode***
- ***uint32_t OffStateIDLEMode***
- ***uint32_t LockLevel***
- ***uint32_t DeadTime***
- ***uint32_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode***
  TIM off state in run mode This parameter can be a value of ***TIM_OSSR_Off_State_Selection_for_Run_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode***
  TIM off state in IDLE mode This parameter can be a value of ***TIM_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel***
  TIM Lock level This parameter can be a value of ***TIM_Lock_level***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime***
  TIM dead Time This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState***
  TIM Break State This parameter can be a value of ***TIM_Break_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity***
  TIM Break input polarity This parameter can be a value of ***TIM_Break_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakFilter***
  Specifies the break input filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2State***
  TIM Break2 State This parameter can be a value of ***TIM_Break2_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Polarity***
  TIM Break2 input polarity This parameter can be a value of ***TIM_Break2_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Filter***
  TIM break2 input filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput***
  TIM Automatic Output Enable state This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

65.1.11 TIM_HandleTypeDef

Data Fields

- ***TIM_TypeDef * Instance***
- ***TIM_Base_InitTypeDef Init***
- ***HAL_TIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***

- ***HAL_LockTypeDef Lock***
- ***__IO HAL_TIM_StateTypeDef State***

Field Documentation

- ***TIM_TypeDef* TIM_HandleTypeDef::Instance***
  Register base address
- ***TIM_Base_InitTypeDef TIM_HandleTypeDef::Init***
  TIM Time Base required parameters
- ***HAL_TIM_ActiveChannel TIM_HandleTypeDef::Channel***
  Active channel
- ***DMA_HandleTypeDef* TIM_HandleTypeDef::hdma[7]***
  DMA Handlers array This array is accessed by a ***DMA_Handle_index***
- ***HAL_LockTypeDef TIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State***
  TIM operation state

65.2 TIM Firmware driver API description

65.2.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
   – Input Capture
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output

65.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   – Time Base: HAL_TIM_Base_MspInit()
   – Input Capture: HAL_TIM_IC_MspInit()
   – Output Compare: HAL_TIM_OC_MspInit()
   – PWM generation: HAL_TIM_PWM_MspInit()
   – One-pulse mode output: HAL_TIM_OnePulse_MspInit()
   – Encoder mode output: HAL_TIM_Encoder_MspInit()
2. Initialize the TIM low level resources:
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      – Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      – Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
   – HAL_TIM_Base_Init: to use the Timer to generate a simple time base

  - HAL_TIM_OC_Init and HAL_TIM_OC_ConfigChannel: to use the Timer to generate an Output Compare signal.
  - HAL_TIM_PWM_Init and HAL_TIM_PWM_ConfigChannel: to use the Timer to generate a PWM signal.
  - HAL_TIM_IC_Init and HAL_TIM_IC_ConfigChannel: to use the Timer to measure an external signal.
  - HAL_TIM_OnePulse_Init and HAL_TIM_OnePulse_ConfigChannel: to use the Timer in One Pulse Mode.
  - HAL_TIM_Encoder_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
  - Time Base: HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_Base_Start_IT()
  - Input Capture: HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_IC_Start_IT()
  - Output Compare: HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_OC_Start_IT()
  - PWM generation: HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_TIM_PWM_Start_IT()
  - One-pulse mode output: HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_Start_IT()
  - Encoder mode output: HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_DMA(), HAL_TIM_Encoder_Start_IT().
6. The DMA Burst is managed with the two following functions: HAL_TIM_DMABurst_WriteStart() HAL_TIM_DMABurst_ReadStart()

65.2.3 Time Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Time Base.
- Stop the Time Base.
- Start the Time Base and enable interrupt.
- Stop the Time Base and disable interrupt.
- Start the Time Base and enable DMA transfer.
- Stop the Time Base and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_Base_Init()***
- ***HAL_TIM_Base_DeInit()***
- ***HAL_TIM_Base_MspInit()***
- ***HAL_TIM_Base_MspDeInit()***
- ***HAL_TIM_Base_Start()***
- ***HAL_TIM_Base_Stop()***
- ***HAL_TIM_Base_Start_IT()***
- ***HAL_TIM_Base_Stop_IT()***
- ***HAL_TIM_Base_Start_DMA()***
- ***HAL_TIM_Base_Stop_DMA()***

65.2.4 Time Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the Time Output Compare.
- Stop the Time Output Compare.
- Start the Time Output Compare and enable interrupt.
- Stop the Time Output Compare and disable interrupt.
- Start the Time Output Compare and enable DMA transfer.
- Stop the Time Output Compare and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_OC_Init()***
- ***HAL_TIM_OC_DeInit()***
- ***HAL_TIM_OC_MspInit()***
- ***HAL_TIM_OC_MspDeInit()***
- ***HAL_TIM_OC_Start()***
- ***HAL_TIM_OC_Stop()***
- ***HAL_TIM_OC_Start_IT()***
- ***HAL_TIM_OC_Stop_IT()***
- ***HAL_TIM_OC_Start_DMA()***
- ***HAL_TIM_OC_Stop_DMA()***

65.2.5 Time PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM OPWM.
- De-initialize the TIM PWM.
- Start the Time PWM.
- Stop the Time PWM.
- Start the Time PWM and enable interrupt.
- Stop the Time PWM and disable interrupt.
- Start the Time PWM and enable DMA transfer.
- Stop the Time PWM and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_PWM_Init()***
- ***HAL_TIM_PWM_DeInit()***
- ***HAL_TIM_PWM_MspInit()***
- ***HAL_TIM_PWM_MspDeInit()***
- ***HAL_TIM_PWM_Start()***
- ***HAL_TIM_PWM_Stop()***
- ***HAL_TIM_PWM_Start_IT()***
- ***HAL_TIM_PWM_Stop_IT()***
- ***HAL_TIM_PWM_Start_DMA()***
- ***HAL_TIM_PWM_Stop_DMA()***

65.2.6 Time Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the Time Input Capture.
- Stop the Time Input Capture.
- Start the Time Input Capture and enable interrupt.
- Stop the Time Input Capture and disable interrupt.
- Start the Time Input Capture and enable DMA transfer.
- Stop the Time Input Capture and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_IC_Init()***
- ***HAL_TIM_IC_DeInit()***
- ***HAL_TIM_IC_MspInit()***
- ***HAL_TIM_IC_MspDeInit()***
- ***HAL_TIM_IC_Start()***
- ***HAL_TIM_IC_Stop()***
- ***HAL_TIM_IC_Start_IT()***
- ***HAL_TIM_IC_Stop_IT()***
- ***HAL_TIM_IC_Start_DMA()***
- ***HAL_TIM_IC_Stop_DMA()***

65.2.7 Time One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the Time One Pulse.
- Stop the Time One Pulse.
- Start the Time One Pulse and enable interrupt.
- Stop the Time One Pulse and disable interrupt.
- Start the Time One Pulse and enable DMA transfer.
- Stop the Time One Pulse and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_OnePulse_Init()***
- ***HAL_TIM_OnePulse_DeInit()***
- ***HAL_TIM_OnePulse_MspInit()***
- ***HAL_TIM_OnePulse_MspDeInit()***
- ***HAL_TIM_OnePulse_Start()***
- ***HAL_TIM_OnePulse_Stop()***
- ***HAL_TIM_OnePulse_Start_IT()***
- ***HAL_TIM_OnePulse_Stop_IT()***

65.2.8 Time Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the Time Encoder.
- Stop the Time Encoder.

- Start the Time Encoder and enable interrupt.
- Stop the Time Encoder and disable interrupt.
- Start the Time Encoder and enable DMA transfer.
- Stop the Time Encoder and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_Encoder_Init()*
- *HAL_TIM_Encoder_DeInit()*
- *HAL_TIM_Encoder_MspInit()*
- *HAL_TIM_Encoder_MspDeInit()*
- *HAL_TIM_Encoder_Start()*
- *HAL_TIM_Encoder_Stop()*
- *HAL_TIM_Encoder_Start_IT()*
- *HAL_TIM_Encoder_Stop_IT()*
- *HAL_TIM_Encoder_Start_DMA()*
- *HAL_TIM_Encoder_Stop_DMA()*

65.2.9 IRQ handler management

This section provides Timer IRQ handler function.

This section contains the following APIs:

- *HAL_TIM_IRQHandler()*

65.2.10 Peripheral Control functions

This section provides functions allowing to:

- Configure The Input Output channels for OC, PWM, IC or One Pulse mode.
- Configure External Clock source.
- Configure Complementary channels, break features and dead time.
- Configure Master and the Slave synchronization.
- Configure the DMA Burst Mode.

This section contains the following APIs:

- *HAL_TIM_OC_ConfigChannel()*
- *HAL_TIM_IC_ConfigChannel()*
- *HAL_TIM_PWM_ConfigChannel()*
- *HAL_TIM_OnePulse_ConfigChannel()*
- *HAL_TIM_DMABurst_WriteStart()*
- *HAL_TIM_DMABurst_WriteStop()*
- *HAL_TIM_DMABurst_ReadStart()*
- *HAL_TIM_DMABurst_ReadStop()*
- *HAL_TIM_GenerateEvent()*
- *HAL_TIM_ConfigOCrefClear()*
- *HAL_TIM_ConfigClockSource()*
- *HAL_TIM_ConfigTI1Input()*
- *HAL_TIM_SlaveConfigSynchronization()*
- *HAL_TIM_SlaveConfigSynchronization_IT()*
- *HAL_TIM_ReadCapturedValue()*

65.2.11 TIM Callbacks functions

This section provides TIM callback functions:

- Timer Period elapsed callback
- Timer Output Compare callback
- Timer Input capture callback
- Timer Trigger callback
- Timer Error callback

This section contains the following APIs:

- ***HAL_TIM_PeriodElapsedCallback()***
- ***HAL_TIM_OC_DelayElapsedCallback()***
- ***HAL_TIM_IC_CaptureCallback()***
- ***HAL_TIM_PWM_PulseFinishedCallback()***
- ***HAL_TIM_TriggerCallback()***
- ***HAL_TIM_ErrorCallback()***

65.2.12 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_TIM_Base_GetState()***
- ***HAL_TIM_OC_GetState()***
- ***HAL_TIM_PWM_GetState()***
- ***HAL_TIM_IC_GetState()***
- ***HAL_TIM_OnePulse_GetState()***
- ***HAL_TIM_Encoder_GetState()***

65.2.13 Detailed description of functions

HAL_TIM_Base_Init

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.
Parameters	• **htim:** TIM Base handle
Return values	• **HAL:** status

HAL_TIM_Base_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize the TIM Base peripheral.
Parameters	• **htim:** TIM Base handle
Return values	• **HAL:** status

HAL_TIM_Base_MspInit

Function name	**void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Base MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_Base_MspDeInit

Function name	**void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize TIM Base MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_Base_Start

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)**
Function description	Starts the TIM Base generation.
Parameters	• **htim:** : TIM handle
Return values	• **HAL:** status

HAL_TIM_Base_Stop

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)**
Function description	Stops the TIM Base generation.
Parameters	• **htim:** : TIM handle
Return values	• **HAL:** status

HAL_TIM_Base_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)**
Function description	Starts the TIM Base generation in interrupt mode.
Parameters	• **htim:** : TIM handle
Return values	• **HAL:** status

HAL_TIM_Base_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)**
Function description	Stops the TIM Base generation in interrupt mode.
Parameters	• **htim:** : TIM handle

Return values	• **HAL:** status

HAL_TIM_Base_Start_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)**
Function description	Starts the TIM Base generation in DMA mode.
Parameters	• **htim:** : TIM handle • **pData:** The source Buffer address. • **Length:** The length of data to be transferred from memory to peripheral.
Return values	• **HAL:** status

HAL_TIM_Base_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)**
Function description	Stops the TIM Base generation in DMA mode.
Parameters	• **htim:** : TIM handle
Return values	• **HAL:** status

HAL_TIM_OC_Init

Function name	**HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.
Parameters	• **htim:** TIM Output Compare handle
Return values	• **HAL:** status

HAL_TIM_OC_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize the TIM peripheral.
Parameters	• **htim:** TIM Output Compare handle
Return values	• **HAL:** status

HAL_TIM_OC_MspInit

Function name	**void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Output Compare MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_OC_MspDeInit

Function name	**void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize TIM Output Compare MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_OC_Start

Function name	**HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Output Compare signal generation.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected – TIM_CHANNEL_5: TIM Channel 5 selected – TIM_CHANNEL_6: TIM Channel 6 selected
Return values	• **HAL:** status

HAL_TIM_OC_Stop

Function name	**HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Output Compare signal generation.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected – TIM_CHANNEL_5: TIM Channel 5 selected – TIM_CHANNEL_6: TIM Channel 6 selected
Return values	• **HAL:** status

HAL_TIM_OC_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Output Compare signal generation in interrupt mode.
Parameters	• **htim:** : TIM OC handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:

- TIM_CHANNEL_1: TIM Channel 1 selected
- TIM_CHANNEL_2: TIM Channel 2 selected
- TIM_CHANNEL_3: TIM Channel 3 selected
- TIM_CHANNEL_4: TIM Channel 4 selected
- TIM_CHANNEL_5: TIM Channel 5 selected
- TIM_CHANNEL_6: TIM Channel 6 selected

Return values
- **HAL:** status

HAL_TIM_OC_Stop_IT

Function name
HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description
Stops the TIM Output Compare signal generation in interrupt mode.

Parameters
- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values
- **HAL:** status

HAL_TIM_OC_Start_DMA

Function name
HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description
Starts the TIM Output Compare signal generation in DMA mode.

Parameters
- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values
- **HAL:** status

HAL_TIM_OC_Stop_DMA

Function name
HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description	Stops the TIM Output Compare signal generation in DMA mode.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected – TIM_CHANNEL_5: TIM Channel 5 selected – TIM_CHANNEL_6: TIM Channel 6 selected
Return values	• **HAL:** status

HAL_TIM_PWM_Init

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.
Parameters	• **htim:** TIM handle
Return values	• **HAL:** status

HAL_TIM_PWM_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize the TIM peripheral.
Parameters	• **htim:** TIM handle
Return values	• **HAL:** status

HAL_TIM_PWM_MspInit

Function name	**void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM PWM MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_PWM_MspDeInit

Function name	**void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize TIM PWM MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_PWM_Start

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the PWM signal generation.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected – TIM_CHANNEL_5: TIM Channel 5 selected – TIM_CHANNEL_6: TIM Channel 6 selected
Return values	• **HAL:** status

HAL_TIM_PWM_Stop

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the PWM signal generation.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected – TIM_CHANNEL_5: TIM Channel 5 selected – TIM_CHANNEL_6: TIM Channel 6 selected
Return values	• **HAL:** status

HAL_TIM_PWM_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the PWM signal generation in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_PWM_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT**

	(TIM_HandleTypeDef * htim, uint32_t Channel)
Function description	Stops the PWM signal generation in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_PWM_Start_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**
Function description	Starts the TIM PWM signal generation in DMA mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected • **pData:** The source Buffer address. • **Length:** The length of data to be transferred from memory to TIM peripheral
Return values	• **HAL:** status

HAL_TIM_PWM_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM PWM signal generation in DMA mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_IC_Init

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Input Capture Time base according to the

specified parameters in the TIM_HandleTypeDef and initialize the associated handle.

Parameters	• **htim:** TIM Input Capture handle
Return values	• **HAL:** status

HAL_TIM_IC_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize the TIM peripheral.
Parameters	• **htim:** TIM Input Capture handle
Return values	• **HAL:** status

HAL_TIM_IC_MspInit

Function name	**void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM INput Capture MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_IC_MspDeInit

Function name	**void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize TIM Input Capture MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_IC_Start

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Input Capture measurement.
Parameters	• **htim:** : TIM Input Capture handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_IC_Stop

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description	Stops the TIM Input Capture measurement.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_IC_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Input Capture measurement in interrupt mode.
Parameters	• **htim:** : TIM Input Capture handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_IC_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Input Capture measurement in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_IC_Start_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**
Function description	Starts the TIM Input Capture measurement on in DMA mode.
Parameters	• **htim:** : TIM Input Capture handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected

	– TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected • **pData:** The destination Buffer address. • **Length:** The length of data to be transferred from TIM peripheral to memory.
Return values	• **HAL:** status

HAL_TIM_IC_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Input Capture measurement in DMA mode.
Parameters	• **htim:** : TIM Input Capture handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIM_OnePulse_Init

Function name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)**
Function description	Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.
Parameters	• **htim:** TIM OnePulse handle • **OnePulseMode:** Select the One pulse mode. This parameter can be one of the following values: – TIM_OPMODE_SINGLE: Only one pulse will be generated. – TIM_OPMODE_REPETITIVE: Repetitive pulses will be generated.
Return values	• **HAL:** status

HAL_TIM_OnePulse_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize the TIM One Pulse.
Parameters	• **htim:** TIM One Pulse handle
Return values	• **HAL:** status

HAL_TIM_OnePulse_MspInit

Function name	**void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM One Pulse MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_OnePulse_MspDeInit

Function name	**void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize TIM One Pulse MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_OnePulse_Start

Function name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function description	Starts the TIM One Pulse signal generation.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• **HAL:** status

HAL_TIM_OnePulse_Stop

Function name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function description	Stops the TIM One Pulse signal generation.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channels to be disable This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• **HAL:** status

HAL_TIM_OnePulse_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function description	Starts the TIM One Pulse signal generation in interrupt mode.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values:

- TIM_CHANNEL_1: TIM Channel 1 selected
- TIM_CHANNEL_2: TIM Channel 2 selected

Return values	• **HAL:** status

HAL_TIM_OnePulse_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function description	Stops the TIM One Pulse signal generation in interrupt mode.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• **HAL:** status

HAL_TIM_Encoder_Init

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)**
Function description	Initializes the TIM Encoder Interface and initialize the associated handle.
Parameters	• **htim:** TIM Encoder Interface handle • **sConfig:** TIM Encoder Interface configuration structure
Return values	• **HAL:** status

HAL_TIM_Encoder_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize the TIM Encoder interface.
Parameters	• **htim:** TIM Encoder handle
Return values	• **HAL:** status

HAL_TIM_Encoder_MspInit

Function name	**void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Encoder Interface MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_Encoder_MspDeInit

Function name	**void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)**

Function description	DeInitialize TIM Encoder Interface MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_Encoder_Start

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Encoder Interface.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• **HAL:** status

HAL_TIM_Encoder_Stop

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Encoder Interface.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• **HAL:** status

HAL_TIM_Encoder_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Encoder Interface in interrupt mode.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• **HAL:** status

HAL_TIM_Encoder_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Encoder Interface in interrupt mode.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• **HAL:** status

HAL_TIM_Encoder_Start_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)**
Function description	Starts the TIM Encoder Interface in DMA mode.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected • **pData1:** The destination Buffer address for IC1. • **pData2:** The destination Buffer address for IC2. • **Length:** The length of data to be transferred from TIM peripheral to memory.
Return values	• **HAL:** status

HAL_TIM_Encoder_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Encoder Interface in DMA mode.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• **HAL:** status

HAL_TIM_IRQHandler

Function name	**void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)**
Function description	This function handles TIM interrupts requests.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIM_OC_ConfigChannel

Function name	**HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)**
Function description	Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	• **htim:** TIM Output Compare handle • **sConfig:** TIM Output Compare configuration structure • **Channel:** : TIM Channels to configure This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected – TIM_CHANNEL_5: TIM Channel 5 selected – TIM_CHANNEL_6: TIM Channel 6 selected
Return values	• **HAL:** status

HAL_TIM_PWM_ConfigChannel

Function name	**HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)**
Function description	Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	• **htim:** TIM PWM handle • **sConfig:** TIM PWM configuration structure • **Channel:** : TIM Channels to be configured This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected – TIM_CHANNEL_5: TIM Channel 5 selected – TIM_CHANNEL_6: TIM Channel 6 selected
Return values	• **HAL:** status

HAL_TIM_IC_ConfigChannel

Function name	**HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig,**

uint32_t Channel)

Function description: Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.

Parameters:
- **htim:** TIM IC handle
- **sConfig:** TIM Input Capture configuration structure
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values:
- **HAL:** status

HAL_TIM_OnePulse_ConfigChannel

Function name: **HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)**

Function description: Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.

Parameters:
- **htim:** TIM One Pulse handle
- **sConfig:** TIM One Pulse configuration structure
- **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
- **InputChannel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values:
- **HAL:** status

HAL_TIM_ConfigOCrefClear

Function name: **HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)**

Function description: Configures the OCRef clear feature.

Parameters:
- **htim:** TIM handle
- **sClearInputConfig:** pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_Channel_1: TIM Channel 1
  - TIM_Channel_2: TIM Channel 2
  - TIM_Channel_3: TIM Channel 3
  - TIM_Channel_4: TIM Channel 4
  - TIM_Channel_5: TIM Channel 5

– TIM_Channel_6: TIM Channel 6

Return values	• **None:**

HAL_TIM_ConfigClockSource

Function name	**HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)**
Function description	Configures the clock source to be used.
Parameters	• **htim:** TIM handle • **sClockSourceConfig:** pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.
Return values	• **HAL:** status

HAL_TIM_ConfigTI1Input

Function name	**HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)**
Function description	Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.
Parameters	• **htim:** TIM handle. • **TI1_Selection:** Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values: – TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 input – TIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)
Return values	• **HAL:** status

HAL_TIM_SlaveConfigSynchronization

Function name	**HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)**
Function description	Configures the TIM in Slave mode.
Parameters	• **htim:** TIM handle. • **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	• **HAL:** status

HAL_TIM_SlaveConfigSynchronization_IT

Function name	**HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)**
Function description	Configures the TIM in Slave mode in interrupt mode.
Parameters	• **htim:** TIM handle. • **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	• **HAL:** status

HAL_TIM_DMABurst_WriteStart

Function name	**HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)**
Function description	Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.
Parameters	• **htim:** TIM handle • **BurstBaseAddress:** TIM Base address from when the DMA will starts the Data write This parameters can be on of the following values: – TIM_DMABASE_CR1 – TIM_DMABASE_CR2 – TIM_DMABASE_SMCR – TIM_DMABASE_DIER – TIM_DMABASE_SR – TIM_DMABASE_EGR – TIM_DMABASE_CCMR1 – TIM_DMABASE_CCMR2 – TIM_DMABASE_CCER – TIM_DMABASE_CNT – TIM_DMABASE_PSC – TIM_DMABASE_ARR – TIM_DMABASE_RCR – TIM_DMABASE_CCR1 – TIM_DMABASE_CCR2 – TIM_DMABASE_CCR3 – TIM_DMABASE_CCR4 – TIM_DMABASE_BDTR – TIM_DMABASE_DCR • **BurstRequestSrc:** TIM DMA Request sources This parameters can be on of the following values: – TIM_DMA_UPDATE: TIM update Interrupt source – TIM_DMA_CC1: TIM Capture Compare 1 DMA source – TIM_DMA_CC2: TIM Capture Compare 2 DMA source

- TIM_DMA_CC3: TIM Capture Compare 3 DMA source
- TIM_DMA_CC4: TIM Capture Compare 4 DMA source
- TIM_DMA_COM: TIM Commutation DMA source
- TIM_DMA_TRIGGER: TIM Trigger DMA source

- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABurstLength_1Transfer and TIM_DMABurstLength_18Transfers.

Return values
- **HAL:** status

HAL_TIM_DMABurst_WriteStop

Function name	**HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)**
Function description	Stops the TIM DMA Burst mode.
Parameters	• **htim:** TIM handle • **BurstRequestSrc:** TIM DMA Request sources to disable
Return values	• **HAL:** status

HAL_TIM_DMABurst_ReadStart

Function name
HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)

Function description
Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters
- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from when the DMA will starts the Data read This parameters can be on of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_DCR

- **BurstRequestSrc:** TIM DMA Request sources This parameters can be on of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABurstLength_1Transfer and TIM_DMABurstLength_18Transfers.

Return values
- **HAL:** status

HAL_TIM_DMABurst_ReadStop

Function name
HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description
Stop the DMA burst reading.

Parameters
- **htim:** TIM handle
- **BurstRequestSrc:** TIM DMA Request sources to disable.

Return values
- **HAL:** status

HAL_TIM_GenerateEvent

Function name
HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)

Function description
Generate a software event.

Parameters
- **htim:** TIM handle
- **EventSource:** specifies the event source. This parameter can be one of the following values:
  - TIM_EVENTSOURCE_UPDATE: Timer update Event source
  - TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source
  - TIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event source
  - TIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event source
  - TIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event source
  - TIM_EVENTSOURCE_COM: Timer COM event source
  - TIM_EVENTSOURCE_TRIGGER: Timer Trigger Event source
  - TIM_EVENTSOURCE_BREAK: Timer Break event source
  - TIM_EVENTSOURCE_BREAK2: Timer Break2 event source

Return values	• **HAL:** status

HAL_TIM_ReadCapturedValue

Function name	**uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Read the captured value from Capture Compare unit.
Parameters	• **htim:** TIM handle. • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **Captured:** value

HAL_TIM_PeriodElapsedCallback

Function name	**void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)**
Function description	Period elapsed callback in non-blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIM_OC_DelayElapsedCallback

Function name	**void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)**
Function description	Output Compare callback in non-blocking mode.
Parameters	• **htim:** : TIM OC handle
Return values	• **None:**

HAL_TIM_IC_CaptureCallback

Function name	**void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)**
Function description	Input Capture callback in non-blocking mode.
Parameters	• **htim:** : TIM IC handle
Return values	• **None:**

HAL_TIM_PWM_PulseFinishedCallback

Function name	**void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)**
Function description	PWM Pulse finished callback in non-blocking mode.
Parameters	• **htim:** : TIM handle

Return values	• **None:**

HAL_TIM_TriggerCallback

Function name	**void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)**
Function description	Hall Trigger detection callback in non-blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIM_ErrorCallback

Function name	**void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)**
Function description	Timer error callback in non-blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIM_Base_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM Base handle state.
Parameters	• **htim:** TIM Base handle
Return values	• **HAL:** state

HAL_TIM_OC_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM OC handle state.
Parameters	• **htim:** TIM Ouput Compare handle
Return values	• **HAL:** state

HAL_TIM_PWM_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM PWM handle state.
Parameters	• **htim:** TIM handle
Return values	• **HAL:** state

HAL_TIM_IC_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM Input Capture handle state.

Parameters	• **htim:** TIM IC handle
Return values	• **HAL:** state

HAL_TIM_OnePulse_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM One Pulse Mode handle state.
Parameters	• **htim:** TIM OPM handle
Return values	• **HAL:** state

HAL_TIM_Encoder_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM Encoder Mode handle state.
Parameters	• **htim:** TIM Encoder handle
Return values	• **HAL:** state

TIM_Base_SetConfig

Function name	**void TIM_Base_SetConfig (TIM_TypeDef * TIMx, TIM_Base_InitTypeDef * Structure)**
Function description	Time Base configuration.
Parameters	• **TIMx:** TIM peripheral • **Structure:** TIM Base configuration structure
Return values	• **None:**

TIM_TI1_SetConfig

Function name	**void TIM_TI1_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ICPolarity, uint32_t TIM_ICSelection, uint32_t TIM_ICFilter)**
Function description	Configure the TI1 as Input.
Parameters	• **TIMx:** to select the TIM peripheral. • **TIM_ICPolarity:** : The Input Polarity. This parameter can be one of the following values: – TIM_ICPolarity_Rising – TIM_ICPolarity_Falling – TIM_ICPolarity_BothEdge • **TIM_ICSelection:** specifies the input to be used. This parameter can be one of the following values: – TIM_ICSelection_DirectTI: TIM Input 1 is selected to be connected to IC1. – TIM_ICSelection_IndirectTI: TIM Input 1 is selected to be connected to IC2. – TIM_ICSelection_TRC: TIM Input 1 is selected to be

connected to TRC.
- **TIM_ICFilter:** Specifies the Input Capture Filter. This parameter must be a value between 0x00 and 0x0F.

Return values
- **None:**

Notes
- TIM_ICFilter and TIM_ICPolarity are not used in INDIRECT mode as TI2FP1 (on channel2 path) is used as the input signal. Therefore CCMR1 must be protected against un-initialized filter and polarity values.

TIM_OC2_SetConfig

Function name	**void TIM_OC2_SetConfig (TIM_TypeDef * TIMx, TIM_OC_InitTypeDef * OC_Config)**
Function description	Time Ouput Compare 2 configuration.
Parameters	• **TIMx:** to select the TIM peripheral • **OC_Config:** The ouput configuration structure
Return values	• **None:**

TIM_ETR_SetConfig

Function name	**void TIM_ETR_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)**
Function description	Configures the TIMx External Trigger (ETR).
Parameters	• **TIMx:** to select the TIM peripheral • **TIM_ExtTRGPrescaler:** The external Trigger Prescaler. This parameter can be one of the following values: – TIM_ETRPRESCALER_DIV1: ETRP Prescaler OFF. – TIM_ETRPRESCALER_DIV2: ETRP frequency divided by 2. – TIM_ETRPRESCALER_DIV4: ETRP frequency divided by 4. – TIM_ETRPRESCALER_DIV8: ETRP frequency divided by 8. • **TIM_ExtTRGPolarity:** The external Trigger Polarity. This parameter can be one of the following values: – TIM_ETRPOLARITY_INVERTED: active low or falling edge active. – TIM_ETRPOLARITY_NONINVERTED: active high or rising edge active. • **ExtTRGFilter:** External Trigger Filter. This parameter must be a value between 0x00 and 0x0F
Return values	• **None:**

TIM_DMADelayPulseCplt

Function name	**void TIM_DMADelayPulseCplt (DMA_HandleTypeDef * hdma)**
Function description	TIM DMA Delay Pulse complete callback.

Parameters	• **hdma:** : pointer to DMA handle.
Return values	• **None:**

TIM_DMAError

Function name	**void TIM_DMAError (DMA_HandleTypeDef * hdma)**
Function description	TIM DMA error callback.
Parameters	• **hdma:** : pointer to DMA handle.
Return values	• **None:**

TIM_DMACaptureCplt

Function name	**void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)**
Function description	TIM DMA Capture complete callback.
Parameters	• **hdma:** : pointer to DMA handle.
Return values	• **None:**

TIM_CCxChannelCmd

Function name	**void TIM_CCxChannelCmd (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ChannelState)**
Function description	Enables or disables the TIM Capture Compare Channel x.
Parameters	• **TIMx:** to select the TIM peripheral • **Channel:** specifies the TIM Channel This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 – TIM_CHANNEL_2: TIM Channel 2 – TIM_CHANNEL_3: TIM Channel 3 – TIM_CHANNEL_4: TIM Channel 4 • **ChannelState:** specifies the TIM Channel CCxE bit new state. This parameter can be: TIM_CCx_ENABLE or TIM_CCx_Disable.
Return values	• **None:**

65.3 TIM Firmware driver defines

65.3.1 TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_ENABLE

TIM_AUTOMATICOUTPUT_DISABLE

TIM Auto-Reload Preload

TIM_AUTORELOAD_PRELOAD_DISABLE	TIMx_ARR register is not buffered
TIM_AUTORELOAD_PRELOAD_ENABLE	TIMx_ARR register is buffered

TIM Break input 2 Enable

TIM_BREAK2_DISABLE

TIM_BREAK2_ENABLE

TIM Break Input 2 Polarity

TIM_BREAK2POLARITY_LOW

TIM_BREAK2POLARITY_HIGH

TIM Break Input Enable

TIM_BREAK_ENABLE

TIM_BREAK_DISABLE

TIM Break Input Polarity

TIM_BREAKPOLARITY_LOW

TIM_BREAKPOLARITY_HIGH

TIM Break System

TIM_BREAK_SYSTEM_ECC	Enables and locks the ECC error signal with Break Input of TIM1/8/15/16/17
TIM_BREAK_SYSTEM_PVD	Enables and locks the PVD connection with TIM1/8/15/16/17 Break Input and also the PVDE and PLS bits of the Power Control Interface
TIM_BREAK_SYSTEM_SRAM2_PARITY_ERROR	Enables and locks the SRAM2_PARITY error signal with Break Input of TIM1/8/15/16/17
TIM_BREAK_SYSTEM_LOCKUP	Enables and locks the LOCKUP output of CortexM4 with Break Input of TIM1/15/16/17

TIM Channel

TIM_CHANNEL_1

TIM_CHANNEL_2

TIM_CHANNEL_3

TIM_CHANNEL_4

TIM_CHANNEL_5

TIM_CHANNEL_6

TIM_CHANNEL_ALL

TIM Clear Input Polarity

TIM_CLEARINPUTPOLARITY_INVERTED	Polarity for ETRx pin
TIM_CLEARINPUTPOLARITY_NONINVERTED	Polarity for ETRx pin

TIM Clear Input Prescaler

TIM_CLEARINPUTPRESCALER_DIV1	No prescaler is used
TIM_CLEARINPUTPRESCALER_DIV2	Prescaler for External ETR pin: Capture

	performed once every 2 events.
TIM_CLEARINPUTPRESCALER_DIV4	Prescaler for External ETR pin: Capture performed once every 4 events.
TIM_CLEARINPUTPRESCALER_DIV8	Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Clear Input Source

TIM_CLEARINPUTSOURCE_ETR

TIM_CLEARINPUTSOURCE_OCREFCLR

TIM_CLEARINPUTSOURCE_NONE

TIM Clock Division

TIM_CLOCKDIVISION_DIV1

TIM_CLOCKDIVISION_DIV2

TIM_CLOCKDIVISION_DIV4

TIM Clock Polarity

TIM_CLOCKPOLARITY_INVERTED	Polarity for ETRx clock sources
TIM_CLOCKPOLARITY_NONINVERTED	Polarity for ETRx clock sources
TIM_CLOCKPOLARITY_RISING	Polarity for TIx clock sources
TIM_CLOCKPOLARITY_FALLING	Polarity for TIx clock sources
TIM_CLOCKPOLARITY_BOTHEDGE	Polarity for TIx clock sources

TIM Clock Prescaler

TIM_CLOCKPRESCALER_DIV1	No prescaler is used
TIM_CLOCKPRESCALER_DIV2	Prescaler for External ETR Clock: Capture performed once every 2 events.
TIM_CLOCKPRESCALER_DIV4	Prescaler for External ETR Clock: Capture performed once every 4 events.
TIM_CLOCKPRESCALER_DIV8	Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_ETRMODE2

TIM_CLOCKSOURCE_INTERNAL

TIM_CLOCKSOURCE_ITR0

TIM_CLOCKSOURCE_ITR1

TIM_CLOCKSOURCE_ITR2

TIM_CLOCKSOURCE_ITR3

TIM_CLOCKSOURCE_TI1ED

TIM_CLOCKSOURCE_TI1

TIM_CLOCKSOURCE_TI2

TIM_CLOCKSOURCE_ETRMODE1

TIM Commutation Source

TIM_COMMUTATION_TRGI

TIM_COMMUTATION_SOFTWARE

TIM Counter Mode

TIM_COUNTERMODE_UP

TIM_COUNTERMODE_DOWN

TIM_COUNTERMODE_CENTERALIGNED1

TIM_COUNTERMODE_CENTERALIGNED2

TIM_COUNTERMODE_CENTERALIGNED3

TIM DMA Base Address

TIM_DMABASE_CR1

TIM_DMABASE_CR2

TIM_DMABASE_SMCR

TIM_DMABASE_DIER

TIM_DMABASE_SR

TIM_DMABASE_EGR

TIM_DMABASE_CCMR1

TIM_DMABASE_CCMR2

TIM_DMABASE_CCER

TIM_DMABASE_CNT

TIM_DMABASE_PSC

TIM_DMABASE_ARR

TIM_DMABASE_RCR

TIM_DMABASE_CCR1

TIM_DMABASE_CCR2

TIM_DMABASE_CCR3

TIM_DMABASE_CCR4

TIM_DMABASE_BDTR

TIM_DMABASE_DCR

TIM_DMABASE_DMAR

TIM_DMABASE_OR1

TIM_DMABASE_CCMR3

TIM_DMABASE_CCR5

TIM_DMABASE_CCR6

TIM_DMABASE_OR2

TIM_DMABASE_OR3

TIM DMA Burst Length

TIM_DMABURSTLENGTH_1TRANSFER

TIM_DMABURSTLENGTH_2TRANSFERS

TIM_DMABURSTLENGTH_3TRANSFERS

TIM_DMABURSTLENGTH_4TRANSFERS

TIM_DMABURSTLENGTH_5TRANSFERS

TIM_DMABURSTLENGTH_6TRANSFERS

TIM_DMABURSTLENGTH_7TRANSFERS

TIM_DMABURSTLENGTH_8TRANSFERS

TIM_DMABURSTLENGTH_9TRANSFERS

TIM_DMABURSTLENGTH_10TRANSFERS

TIM_DMABURSTLENGTH_11TRANSFERS

TIM_DMABURSTLENGTH_12TRANSFERS

TIM_DMABURSTLENGTH_13TRANSFERS

TIM_DMABURSTLENGTH_14TRANSFERS

TIM_DMABURSTLENGTH_15TRANSFERS

TIM_DMABURSTLENGTH_16TRANSFERS

TIM_DMABURSTLENGTH_17TRANSFERS

TIM_DMABURSTLENGTH_18TRANSFERS

TIM DMA Sources

TIM_DMA_UPDATE

TIM_DMA_CC1

TIM_DMA_CC2

TIM_DMA_CC3

TIM_DMA_CC4

TIM_DMA_COM

TIM_DMA_TRIGGER

TIM Encoder Mode

TIM_ENCODERMODE_TI1

TIM_ENCODERMODE_TI2

TIM_ENCODERMODE_TI12

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED	Polarity for ETR source
TIM_ETRPOLARITY_NONINVERTED	Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1	No prescaler is used

TIM_ETRPRESCALER_DIV2	ETR input source is divided by 2
TIM_ETRPRESCALER_DIV4	ETR input source is divided by 4
TIM_ETRPRESCALER_DIV8	ETR input source is divided by 8

TIM Extended Event Source

TIM_EVENTSOURCE_UPDATE	Reinitialize the counter and generates an update of the registers
TIM_EVENTSOURCE_CC1	A capture/compare event is generated on channel 1
TIM_EVENTSOURCE_CC2	A capture/compare event is generated on channel 2
TIM_EVENTSOURCE_CC3	A capture/compare event is generated on channel 3
TIM_EVENTSOURCE_CC4	A capture/compare event is generated on channel 4
TIM_EVENTSOURCE_COM	A commutation event is generated
TIM_EVENTSOURCE_TRIGGER	A trigger event is generated
TIM_EVENTSOURCE_BREAK	A break event is generated
TIM_EVENTSOURCE_BREAK2	A break 2 event is generated

TIM Exported Macros

__HAL_TIM_RESET_HANDLE_STATE

Description:

- Reset TIM handle state.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

__HAL_TIM_ENABLE

Description:

- Enable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_ENABLE

Description:

- Enable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_DISABLE

Description:

- Disable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_DISABLE

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled only if all the CCx and CCxN channels have been disabled

__HAL_TIM_MOE_DISABLE_UNCONDITIONALLY

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled unconditionally

__HAL_TIM_ENABLE_IT

Description:

- Enable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt

Return value:

- None

__HAL_TIM_DISABLE_IT

Description:

- Disable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt

Return value:

- None

__HAL_TIM_ENABLE_DMA

Description:

- Enable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to enable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_DISABLE_DMA

Description:

- Disable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to disable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_GET_FLAG

Description:

- Check whether the specified TIM interrupt flag is set or not.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag

- TIM_FLAG_BREAK2: Break 2 interrupt flag
- TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
- TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
- TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
- TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
- TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_CLEAR_FLAG

Description:

- Clear the specified TIM interrupt flag.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF:

Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_GET_IT_SOURCE

Description:

- Check whether the specified TIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the TIM interrupt source to check. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt

Return value:

- The: state of TIM_IT (SET or RESET).

__HAL_TIM_CLEAR_IT

Description:

- Clear the TIM interrupt pending bits.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt

Return value:

- None

__HAL_TIM_IS_TIM_COUNTING_DOWN

Description:

- Indicates whether or not the TIM Counter is used as downcounter.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- False: (Counter used as upcounter) or True (Counter used as downcounter)

Notes:

- This macro is particularly useful to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

__HAL_TIM_SET_PRESCALER

Description:

- Set the TIM Prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __PRESC__: specifies the Prescaler new value.

Return value:

- None

__HAL_TIM_SET_COUNTER

Description:

- Set the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __COUNTER__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_COUNTER

Description:

- Get the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer counter register (TIMx_CNT)

__HAL_TIM_SET_AUTORELOAD

Description:

- Set the TIM Autoreload Register value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __AUTORELOAD__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_AUTORELOAD

Description:

- Get the TIM Autoreload Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer auto-reload register(TIMx_ARR)

__HAL_TIM_SET_CLOCKDIVISION

Description:

- Set the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __CKD__: specifies the clock division value. This parameter can be one of the following value:
  - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  - TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  - TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

Return value:

- None

__HAL_TIM_GET_CLOCKDIVISION

Description:

- Get the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- The: clock division can be one of the

following values:
- TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
- TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
- TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

__HAL_TIM_SET_ICPRESCALER

Description:

- Set the TIM Input Capture prescaler on runtime without calling another time

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __ICPSC__: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

Return value:

- None

__HAL_TIM_GET_ICPRESCALER

Description:

- Get the TIM Input Capture prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: get input capture 1 prescaler value
  - TIM_CHANNEL_2: get input capture 2 prescaler value
  - TIM_CHANNEL_3: get input capture 3 prescaler value

- TIM_CHANNEL_4: get input capture 4 prescaler value

Return value:

- The: input capture prescaler can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

__HAL_TIM_SET_COMPARE

Description:

- Set the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- __COMPARE__: specifies the Capture Compare register new value.

Return value:

- None

__HAL_TIM_GET_COMPARE

Description:

- Get the TIM Capture Compare Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channel associated with the capture compare register This parameter can be one of the following values:
  - TIM_CHANNEL_1: get capture/compare 1 register value

  - TIM_CHANNEL_2: get capture/compare 2 register value
  - TIM_CHANNEL_3: get capture/compare 3 register value
  - TIM_CHANNEL_4: get capture/compare 4 register value
  - TIM_CHANNEL_5: get capture/compare 5 register value
  - TIM_CHANNEL_6: get capture/compare 6 register value

Return value:

- 16-bit: or 32-bit value of the capture/compare register (TIMx_CCRy)

__HAL_TIM_ENABLE_OCxPRELOAD

Description:

- Set the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_DISABLE_OCxPRELOAD

Description:

- Reset the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4

selected
- TIM_CHANNEL_5: TIM Channel 5 selected
- TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_URS_ENABLE

Description:

- Set the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the USR bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

__HAL_TIM_URS_DISABLE

Description:

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the USR bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): _ Counter overflow underflow _ Setting the UG bit _ Update generation through the slave mode controller

__HAL_TIM_SET_CAPTURE POLARITY

Description:

- Set the TIM Capture x input polarity on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1

selected
- TIM_CHANNEL_2: TIM Channel 2 selected
- TIM_CHANNEL_3: TIM Channel 3 selected
- TIM_CHANNEL_4: TIM Channel 4 selected

- __POLARITY__: Polarity for TIx source
  - TIM_INPUTCHANNELPOLARITY_RISING: Rising Edge
  - TIM_INPUTCHANNELPOLARITY_FALLING: Falling Edge
  - TIM_INPUTCHANNELPOLARITY_BOTHEDGE: Rising and Falling Edge

Return value:

- None

TIM Flag Definition

TIM_FLAG_UPDATE

TIM_FLAG_CC1

TIM_FLAG_CC2

TIM_FLAG_CC3

TIM_FLAG_CC4

TIM_FLAG_CC5

TIM_FLAG_CC6

TIM_FLAG_COM

TIM_FLAG_TRIGGER

TIM_FLAG_BREAK

TIM_FLAG_BREAK2

TIM_FLAG_SYSTEM_BREAK

TIM_FLAG_CC1OF

TIM_FLAG_CC2OF

TIM_FLAG_CC3OF

TIM_FLAG_CC4OF

Group Channel 5 and Channel 1, 2 or 3

TIM_GROUPCH5_NONE

TIM_GROUPCH5_OC1REFC

TIM_GROUPCH5_OC2REFC

TIM_GROUPCH5_OC3REFC

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING

TIM_ICPOLARITY_FALLING

TIM_ICPOLARITY_BOTHEDGE

TIM Input Capture Prescaler

TIM_ICPSC_DIV1	Capture performed each time an edge is detected on the capture input
TIM_ICPSC_DIV2	Capture performed once every 2 events
TIM_ICPSC_DIV4	Capture performed once every 4 events
TIM_ICPSC_DIV8	Capture performed once every 8 events

TIM Input Capture Selection

TIM_ICSELECTION_DIRECTTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively
TIM_ICSELECTION_INDIRECTTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively
TIM_ICSELECTION_TRC	TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

TIM Input Channel polarity

TIM_INPUTCHANNELPOLARITY_RISING	Polarity for TIx source
TIM_INPUTCHANNELPOLARITY_FALLING	Polarity for TIx source
TIM_INPUTCHANNELPOLARITY_BOTHEDGE	Polarity for TIx source

TIM interrupt Definition

TIM_IT_UPDATE

TIM_IT_CC1

TIM_IT_CC2

TIM_IT_CC3

TIM_IT_CC4

TIM_IT_COM

TIM_IT_TRIGGER

TIM_IT_BREAK

TIM Lock level

TIM_LOCKLEVEL_OFF

TIM_LOCKLEVEL_1

TIM_LOCKLEVEL_2

TIM_LOCKLEVEL_3

TIM Master Mode Selection

TIM_TRGO_RESET

TIM_TRGO_ENABLE

TIM_TRGO_UPDATE

TIM_TRGO_OC1

TIM_TRGO_OC1REF

TIM_TRGO_OC2REF

TIM_TRGO_OC3REF

TIM_TRGO_OC4REF

TIM Master Mode Selection 2 (TRGO2)

TIM_TRGO2_RESET

TIM_TRGO2_ENABLE

TIM_TRGO2_UPDATE

TIM_TRGO2_OC1

TIM_TRGO2_OC1REF

TIM_TRGO2_OC2REF

TIM_TRGO2_OC3REF

TIM_TRGO2_OC4REF

TIM_TRGO2_OC5REF

TIM_TRGO2_OC6REF

TIM_TRGO2_OC4REF_RISINGFALLING

TIM_TRGO2_OC6REF_RISINGFALLING

TIM_TRGO2_OC4REF_RISING_OC6REF_RISING

TIM_TRGO2_OC4REF_RISING_OC6REF_FALLING

TIM_TRGO2_OC5REF_RISING_OC6REF_RISING

TIM_TRGO2_OC5REF_RISING_OC6REF_FALLING

TIM Master/Slave Mode

TIM_MASTERSLAVEMODE_ENABLE

TIM_MASTERSLAVEMODE_DISABLE

TIM One Pulse Mode

TIM_OPMODE_SINGLE

TIM_OPMODE_REPETITIVE

TIM OSSI OffState Selection for Idle mode state

TIM_OSSI_ENABLE

TIM_OSSI_DISABLE

TIM OSSR OffState Selection for Run mode state

TIM_OSSR_ENABLE

TIM_OSSR_DISABLE

TIM Output Compare and PWM Modes

TIM_OCMODE_TIMING

TIM_OCMODE_ACTIVE

TIM_OCMODE_INACTIVE

TIM_OCMODE_TOGGLE

TIM_OCMODE_PWM1

TIM_OCMODE_PWM2

TIM_OCMODE_FORCED_ACTIVE

TIM_OCMODE_FORCED_INACTIVE

TIM_OCMODE_RETRIGERRABLE_OPM1

TIM_OCMODE_RETRIGERRABLE_OPM2

TIM_OCMODE_COMBINED_PWM1

TIM_OCMODE_COMBINED_PWM2

TIM_OCMODE_ASSYMETRIC_PWM1

TIM_OCMODE_ASSYMETRIC_PWM2

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

TIM_OCIDLESTATE_RESET

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

TIM_OCNIDLESTATE_RESET

TIM Complementary Output Compare Polarity

TIM_OCNPOLARITY_HIGH

TIM_OCNPOLARITY_LOW

TIM Complementary Output Compare State

TIM_OUTPUTNSTATE_DISABLE

TIM_OUTPUTNSTATE_ENABLE

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

TIM Output Compare State

TIM_OUTPUTSTATE_DISABLE

TIM_OUTPUTSTATE_ENABLE

TIM Output Fast State

TIM_OCFAST_DISABLE

TIM_OCFAST_ENABLE

TIM Slave mode

TIM_SLAVEMODE_DISABLE

TIM_SLAVEMODE_RESET

TIM_SLAVEMODE_GATED

TIM_SLAVEMODE_TRIGGER

TIM_SLAVEMODE_EXTERNAL1

TIM_SLAVEMODE_COMBINED_RESETTRIGGER

TIM TI1 Input Selection

TIM_TI1SELECTION_CH1

TIM_TI1SELECTION_XORCOMBINATION

TIM Trigger Polarity

TIM_TRIGGERPOLARITY_INVERTED	Polarity for ETRx trigger sources
TIM_TRIGGERPOLARITY_NONINVERTED	Polarity for ETRx trigger sources
TIM_TRIGGERPOLARITY_RISING	Polarity for TIxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_FALLING	Polarity for TIxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_BOTHEDGE	Polarity for TIxFPx or TI1_ED trigger sources

TIM Trigger Prescaler

TIM_TRIGGERPRESCALER_DIV1	No prescaler is used
TIM_TRIGGERPRESCALER_DIV2	Prescaler for External ETR Trigger: Capture performed once every 2 events.
TIM_TRIGGERPRESCALER_DIV4	Prescaler for External ETR Trigger: Capture performed once every 4 events.
TIM_TRIGGERPRESCALER_DIV8	Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection

TIM_TS_ITR0

TIM_TS_ITR1

TIM_TS_ITR2

TIM_TS_ITR3

TIM_TS_TI1F_ED

TIM_TS_TI1FP1

TIM_TS_TI2FP2

TIM_TS_ETRF

TIM_TS_NONE

66 HAL TIM Extension Driver

66.1 TIMEx Firmware driver registers structures

66.1.1 TIM_HallSensor_InitTypeDef

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t Commutation_Delay***

Field Documentation

- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_HallSensor_InitTypeDef::Commutation_Delay***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

66.1.2 TIMEx_BreakInputConfigTypeDef

Data Fields

- ***uint32_t Source***
- ***uint32_t Enable***
- ***uint32_t Polarity***

Field Documentation

- ***uint32_t TIMEx_BreakInputConfigTypeDef::Source***
  Specifies the source of the timer break input. This parameter can be a value of ***TIMEx_Break_Input_Source***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Enable***
  Specifies whether or not the break input source is enabled. This parameter can be a value of ***TIMEx_Break_Input_Source_Enable***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Polarity***
  Specifies the break input source polarity. This parameter can be a value of ***TIMEx_Break_Input_Source_Polarity*** Not relevant when analog watchdog output of the DFSDM1 used as break input source

66.2 TIMEx Firmware driver API description

66.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for:

- Output Compare
- PWM generation (Edge and Center-aligned Mode)
- One-pulse mode output

2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

66.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   - Hall Sensor output: HAL_TIMEx_HallSensor_MspInit()
2. Initialize the TIM low level resources:
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      - Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
   - HAL_TIMEx_HallSensor_Init() and HAL_TIMEx_ConfigCommutationEvent(): to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).
5. Activate the TIM peripheral using one of the start functions:
   - Complementary Output Compare: HAL_TIMEx_OCN_Start(), HAL_TIMEx_OCN_Start_DMA(), HAL_TIMEx_OC_Start_IT()
   - Complementary PWM generation: HAL_TIMEx_PWMN_Start(), HAL_TIMEx_PWMN_Start_DMA(), HAL_TIMEx_PWMN_Start_IT()
   - Complementary One-pulse mode output: HAL_TIMEx_OnePulseN_Start(), HAL_TIMEx_OnePulseN_Start_IT()
   - Hall Sensor output: HAL_TIMEx_HallSensor_Start(), HAL_TIMEx_HallSensor_Start_DMA(), HAL_TIMEx_HallSensor_Start_IT().

66.2.3 Timer Hall Sensor functions

This section provides functions allowing to:

- Initialize and configure TIM HAL Sensor.
- De-initialize TIM HAL Sensor.
- Start the Hall Sensor Interface.
- Stop the Hall Sensor Interface.
- Start the Hall Sensor Interface and enable interrupts.
- Stop the Hall Sensor Interface and disable interrupts.
- Start the Hall Sensor Interface and enable DMA transfers.
- Stop the Hall Sensor Interface and disable DMA transfers.

This section contains the following APIs:

- ***HAL_TIMEx_HallSensor_Init()***

- *HAL_TIMEx_HallSensor_DeInit()*
- *HAL_TIMEx_HallSensor_MspInit()*
- *HAL_TIMEx_HallSensor_MspDeInit()*
- *HAL_TIMEx_HallSensor_Start()*
- *HAL_TIMEx_HallSensor_Stop()*
- *HAL_TIMEx_HallSensor_Start_IT()*
- *HAL_TIMEx_HallSensor_Stop_IT()*
- *HAL_TIMEx_HallSensor_Start_DMA()*
- *HAL_TIMEx_HallSensor_Stop_DMA()*

66.2.4 Timer Complementary Output Compare functions

This section provides functions allowing to:

- Start the Complementary Output Compare/PWM.
- Stop the Complementary Output Compare/PWM.
- Start the Complementary Output Compare/PWM and enable interrupts.
- Stop the Complementary Output Compare/PWM and disable interrupts.
- Start the Complementary Output Compare/PWM and enable DMA transfers.
- Stop the Complementary Output Compare/PWM and disable DMA transfers.

This section contains the following APIs:

- *HAL_TIMEx_OCN_Start()*
- *HAL_TIMEx_OCN_Stop()*
- *HAL_TIMEx_OCN_Start_IT()*
- *HAL_TIMEx_OCN_Stop_IT()*
- *HAL_TIMEx_OCN_Start_DMA()*
- *HAL_TIMEx_OCN_Stop_DMA()*

66.2.5 Timer Complementary PWM functions

This section provides functions allowing to:

- Start the Complementary PWM.
- Stop the Complementary PWM.
- Start the Complementary PWM and enable interrupts.
- Stop the Complementary PWM and disable interrupts.
- Start the Complementary PWM and enable DMA transfers.
- Stop the Complementary PWM and disable DMA transfers.
- Start the Complementary Input Capture measurement.
- Stop the Complementary Input Capture.
- Start the Complementary Input Capture and enable interrupts.
- Stop the Complementary Input Capture and disable interrupts.
- Start the Complementary Input Capture and enable DMA transfers.
- Stop the Complementary Input Capture and disable DMA transfers.
- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- *HAL_TIMEx_PWMN_Start()*
- *HAL_TIMEx_PWMN_Stop()*
- *HAL_TIMEx_PWMN_Start_IT()*
- *HAL_TIMEx_PWMN_Stop_IT()*

- *HAL_TIMEx_PWMN_Start_DMA()*
- *HAL_TIMEx_PWMN_Stop_DMA()*

66.2.6 Timer Complementary One Pulse functions

This section provides functions allowing to:

- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- *HAL_TIMEx_OnePulseN_Start()*
- *HAL_TIMEx_OnePulseN_Stop()*
- *HAL_TIMEx_OnePulseN_Start_IT()*
- *HAL_TIMEx_OnePulseN_Stop_IT()*

66.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Output channels for OC and PWM mode.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- Configure timer remapping capabilities.
- Enable or disable channel grouping

This section contains the following APIs:

- *HAL_TIMEx_ConfigCommutationEvent()*
- *HAL_TIMEx_ConfigCommutationEvent_IT()*
- *HAL_TIMEx_ConfigCommutationEvent_DMA()*
- *HAL_TIMEx_MasterConfigSynchronization()*
- *HAL_TIMEx_ConfigBreakDeadTime()*
- *HAL_TIMEx_ConfigBreakInput()*
- *HAL_TIMEx_RemapConfig()*
- *HAL_TIMEx_GroupChannel5()*

66.2.8 Extended Callbacks functions

This section provides Extended TIM callback functions:

- Timer Commutation callback
- Timer Break callback

This section contains the following APIs:

- *HAL_TIMEx_CommutationCallback()*
- *HAL_TIMEx_BreakCallback()*

66.2.9 Extended Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_TIMEx_HallSensor_GetState()*

66.2.10 Detailed description of functions

HAL_TIMEx_HallSensor_Init

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (TIM_HandleTypeDef * htim, TIM_HallSensor_InitTypeDef * sConfig)**
Function description	Initializes the TIM Hall Sensor Interface and initialize the associated handle.
Parameters	• **htim:** TIM Encoder Interface handle • **sConfig:** TIM Hall Sensor configuration structure
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_DeInit

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize the TIM Hall Sensor interface.
Parameters	• **htim:** TIM Hall Sensor handle
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_MspInit

Function name	**void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Hall Sensor MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIMEx_HallSensor_MspDeInit

Function name	**void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitialize TIM Hall Sensor MSP.
Parameters	• **htim:** TIM handle
Return values	• **None:**

HAL_TIMEx_HallSensor_Start

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)**
Function description	Starts the TIM Hall Sensor Interface.
Parameters	• **htim:** : TIM Hall Sensor handle
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_Stop

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)**
Function description	Stops the TIM Hall sensor Interface.
Parameters	• **htim:** : TIM Hall Sensor handle
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)**
Function description	Starts the TIM Hall Sensor Interface in interrupt mode.
Parameters	• **htim:** : TIM Hall Sensor handle
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)**
Function description	Stops the TIM Hall Sensor Interface in interrupt mode.
Parameters	• **htim:** : TIM handle
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_Start_DMA

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)**
Function description	Starts the TIM Hall Sensor Interface in DMA mode.
Parameters	• **htim:** : TIM Hall Sensor handle • **pData:** The destination Buffer address. • **Length:** The length of data to be transferred from TIM peripheral to memory.
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)**
Function description	Stops the TIM Hall Sensor Interface in DMA mode.
Parameters	• **htim:** : TIM handle
Return values	• **HAL:** status

HAL_TIMEx_OCN_Start

Function name	**HAL_StatusTypeDef HAL_TIMEx_OCN_Start**

	(TIM_HandleTypeDef * htim, uint32_t Channel)
Function description	Starts the TIM Output Compare signal generation on the complementary output.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIMEx_OCN_Stop

Function name	**HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Output Compare signal generation on the complementary output.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIMEx_OCN_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.
Parameters	• **htim:** : TIM OC handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIMEx_OCN_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIMEx_OCN_Start_DMA

Function name	**HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**
Function description	Starts the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected • **pData:** The source Buffer address. • **Length:** The length of data to be transferred from memory to TIM peripheral
Return values	• **HAL:** status

HAL_TIMEx_OCN_Stop_DMA

Function name	**HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIMEx_PWMN_Start

Function name	**HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the PWM signal generation on the complementary output.

Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIMEx_PWMN_Stop

Function name	**HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the PWM signal generation on the complementary output.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIMEx_PWMN_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Starts the PWM signal generation in interrupt mode on the complementary output.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected – TIM_CHANNEL_3: TIM Channel 3 selected – TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• **HAL:** status

HAL_TIMEx_PWMN_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Stops the PWM signal generation in interrupt mode on the complementary output.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected

- TIM_CHANNEL_3: TIM Channel 3 selected
- TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIMEx_PWMN_Start_DMA

Function name
: **HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**

Function description
: Starts the TIM PWM signal generation in DMA mode on the complementary output.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values
- **HAL:** status

HAL_TIMEx_PWMN_Stop_DMA

Function name
: **HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description
: Stops the TIM PWM signal generation in DMA mode on the complementary output.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIMEx_OnePulseN_Start

Function name
: **HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function description
: Starts the TIM One Pulse signal generation on the complementary output.

Parameters
- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected

– TIM_CHANNEL_2: TIM Channel 2 selected

Return values
- **HAL:** status

HAL_TIMEx_OnePulseN_Stop

Function name	**HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function description	Stops the TIM One Pulse signal generation on the complementary output.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channel to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• **HAL:** status

HAL_TIMEx_OnePulseN_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function description	Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channel to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• **HAL:** status

HAL_TIMEx_OnePulseN_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function description	Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channel to be disabled This parameter can be one of the following values: – TIM_CHANNEL_1: TIM Channel 1 selected – TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• **HAL:** status

HAL_TIMEx_ConfigCommutationEvent

Function name	**HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)**

Function description	Configure the TIM commutation event sequence.
Parameters	• **htim:** TIM handle • **InputTrigger:** : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values: – TIM_TS_ITR0: Internal trigger 0 selected – TIM_TS_ITR1: Internal trigger 1 selected – TIM_TS_ITR2: Internal trigger 2 selected – TIM_TS_ITR3: Internal trigger 3 selected – TIM_TS_NONE: No trigger is needed • **CommutationSource:** : the Commutation Event source This parameter can be one of the following values: – TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer – TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
Return values	• **HAL:** status
Notes	• This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutationEvent_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)**
Function description	Configure the TIM commutation event sequence with interrupt.
Parameters	• **htim:** TIM handle • **InputTrigger:** : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values: – TIM_TS_ITR0: Internal trigger 0 selected – TIM_TS_ITR1: Internal trigger 1 selected – TIM_TS_ITR2: Internal trigger 2 selected – TIM_TS_ITR3: Internal trigger 3 selected – TIM_TS_NONE: No trigger is needed • **CommutationSource:** : the Commutation Event source This parameter can be one of the following values: – TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer – TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
Return values	• **HAL:** status
Notes	• This function is mandatory to use the commutation event in

order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutationEvent_DMA

Function name	**HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)**
Function description	Configure the TIM commutation event sequence with DMA.
Parameters	• **htim:** TIM handle • **InputTrigger:** : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values: – TIM_TS_ITR0: Internal trigger 0 selected – TIM_TS_ITR1: Internal trigger 1 selected – TIM_TS_ITR2: Internal trigger 2 selected – TIM_TS_ITR3: Internal trigger 3 selected – TIM_TS_NONE: No trigger is needed • **CommutationSource:** : the Commutation Event source This parameter can be one of the following values: – TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer – TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
Return values	• **HAL:** status
Notes	• This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1. • The user should configure the DMA in his own software, in This function only the COMDE bit is set

HAL_TIMEx_MasterConfigSynchronization

Function name	**HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)**
Function description	Configures the TIM in master mode.
Parameters	• **htim:** TIM handle. • **sMasterConfig:** pointer to a TIM_MasterConfigTypeDef

structure that contains the selected trigger output (TRGO) and the Master/Slave mode.

Return values	• **HAL:** status

HAL_TIMEx_ConfigBreakDeadTime

Function name	**HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)**
Function description	Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).
Parameters	• **htim:** TIM handle • **sBreakDeadTimeConfig:** pointer to a TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.
Return values	• **HAL:** status

HAL_TIMEx_ConfigBreakInput

Function name	**HAL_StatusTypeDef HAL_TIMEx_ConfigBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput, TIMEx_BreakInputConfigTypeDef * sBreakInputConfig)**
Function description	Configures the break input source.
Parameters	• **htim:** TIM handle. • **BreakInput:** Break input to configure This parameter can be one of the following values: – TIM_BREAKINPUT_BRK: Timer break input – TIM_BREAKINPUT_BRK2: Timer break 2 input • **sBreakInputConfig:** Break input source configuration
Return values	• **HAL:** status

HAL_TIMEx_GroupChannel5

Function name	**HAL_StatusTypeDef HAL_TIMEx_GroupChannel5 (TIM_HandleTypeDef * htim, uint32_t Channels)**
Function description	Group channel 5 and channel 1, 2 or 3.
Parameters	• **htim:** TIM handle. • **Channels:** specifies the reference signal(s) the OC5REF is combined with. This parameter can be any combination of the following values: TIM_GROUPCH5_NONE: No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC TIM_GROUPCH5_OC1REFC: OC1REFC is the logical AND of OC1REFC and OC5REF TIM_GROUPCH5_OC2REFC: OC2REFC is the logical AND of OC2REFC and OC5REF TIM_GROUPCH5_OC3REFC: OC3REFC is the logical AND of OC3REFC and OC5REF
Return values	• **HAL:** status

HAL_TIMEx_RemapConfig

Function name	**HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)**
Function description	Configures the TIMx Remapping input capabilities.
Parameters	• **htim:** TIM handle. • **Remap:** specifies the TIM remapping source.
Return values	• **HAL:** status

HAL_TIMEx_CommutationCallback

Function name	**void HAL_TIMEx_CommutationCallback (TIM_HandleTypeDef * htim)**
Function description	Hall commutation changed callback in non-blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIMEx_BreakCallback

Function name	**void HAL_TIMEx_BreakCallback (TIM_HandleTypeDef * htim)**
Function description	Hall Break detection callback in non-blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIMEx_HallSensor_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM Hall Sensor interface handle state.
Parameters	• **htim:** TIM Hall Sensor handle
Return values	• **HAL:** state

TIMEx_DMACommutationCplt

Function name	**void TIMEx_DMACommutationCplt (DMA_HandleTypeDef * hdma)**
Function description	TIM DMA Commutation callback.
Parameters	• **hdma:** : pointer to DMA handle.
Return values	• **None:**

66.3 TIMEx Firmware driver defines

66.3.1 TIMEx

TIM Extended Break input

TIM_BREAKINPUT_BRK

TIM_BREAKINPUT_BRK2

TIM Extended Break input source

TIM_BREAKINPUTSOURCE_BKIN

TIM_BREAKINPUTSOURCE_COMP1

TIM_BREAKINPUTSOURCE_COMP2

TIM_BREAKINPUTSOURCE_DFSDM1

TIM Extended Break input source enabling

TIM_BREAKINPUTSOURCE_DISABLE

TIM_BREAKINPUTSOURCE_ENABLE

TIM Extended Break input polarity

TIM_BREAKINPUTSOURCE_POLARITY_LOW

TIM_BREAKINPUTSOURCE_POLARITY_HIGH

TIM Extended Remapping

TIM_TIM1_ETR_ADC1_NONE

TIM_TIM1_ETR_ADC1_AWD1

TIM_TIM1_ETR_ADC1_AWD2

TIM_TIM1_ETR_ADC1_AWD3

TIM_TIM1_TI1_GPIO

TIM_TIM1_TI1_COMP1

TIM_TIM1_ETR_GPIO

TIM_TIM1_ETR_COMP1

TIM_TIM1_ETR_COMP2

TIM_TIM2_ITR1_TIM8_TRGO

TIM_TIM2_ITR1_OTG_FS_SOF

TIM_TIM2_ETR_GPIO

TIM_TIM2_ETR_LSE

TIM_TIM2_ETR_COMP1

TIM_TIM2_ETR_COMP2

TIM_TIM2_TI4_GPIO

TIM_TIM2_TI4_COMP1

TIM_TIM2_TI4_COMP2

TIM_TIM2_TI4_COMP1_COMP2

TIM_TIM3_TI1_GPIO

TIM_TIM3_TI1_COMP1

TIM_TIM3_TI1_COMP2

TIM_TIM3_TI1_COMP1_COMP2

TIM_TIM3_ETR_GPIO

TIM_TIM3_ETR_COMP1

TIM_TIM8_TI1_GPIO

TIM_TIM8_TI1_COMP2

TIM_TIM8_ETR_GPIO

TIM_TIM8_ETR_COMP1

TIM_TIM8_ETR_COMP2

TIM_TIM15_TI1_GPIO

TIM_TIM15_TI1_LSE

TIM_TIM15_ENCODERMODE_NONE

TIM_TIM15_ENCODERMODE_TIM2

TIM_TIM15_ENCODERMODE_TIM3

TIM_TIM15_ENCODERMODE_TIM4

TIM_TIM16_TI1_GPIO

TIM_TIM16_TI1_LSI

TIM_TIM16_TI1_LSE

TIM_TIM16_TI1_RTC

TIM_TIM17_TI1_GPIO

TIM_TIM17_TI1_MSI

TIM_TIM17_TI1_HSE_32

TIM_TIM17_TI1_MCO

67 HAL TSC Generic Driver

67.1 TSC Firmware driver registers structures

67.1.1 TSC_InitTypeDef

Data Fields

- ***uint32_t CTPulseHighLength***
- ***uint32_t CTPulseLowLength***
- ***uint32_t SpreadSpectrum***
- ***uint32_t SpreadSpectrumDeviation***
- ***uint32_t SpreadSpectrumPrescaler***
- ***uint32_t PulseGeneratorPrescaler***
- ***uint32_t MaxCountValue***
- ***uint32_t IODefaultMode***
- ***uint32_t SynchroPinPolarity***
- ***uint32_t AcquisitionMode***
- ***uint32_t MaxCountInterrupt***
- ***uint32_t ChannelIOs***
- ***uint32_t ShieldIOs***
- ***uint32_t SamplingIOs***

Field Documentation

- ***uint32_t TSC_InitTypeDef::CTPulseHighLength***
  Charge-transfer high pulse length This parameter can be a value of ***TSC_CTPulseHL_Config***
- ***uint32_t TSC_InitTypeDef::CTPulseLowLength***
  Charge-transfer low pulse length This parameter can be a value of ***TSC_CTPulseLL_Config***
- ***uint32_t TSC_InitTypeDef::SpreadSpectrum***
  Spread spectrum activation This parameter can be a value of ***TSC_CTPulseLL_Config***
- ***uint32_t TSC_InitTypeDef::SpreadSpectrumDeviation***
  Spread spectrum deviation This parameter must be a number between Min_Data = 0 and Max_Data = 127
- ***uint32_t TSC_InitTypeDef::SpreadSpectrumPrescaler***
  Spread spectrum prescaler This parameter can be a value of ***TSC_SpreadSpec_Prescaler***
- ***uint32_t TSC_InitTypeDef::PulseGeneratorPrescaler***
  Pulse generator prescaler This parameter can be a value of ***TSC_PulseGenerator_Prescaler***
- ***uint32_t TSC_InitTypeDef::MaxCountValue***
  Max count value This parameter can be a value of ***TSC_MaxCount_Value***
- ***uint32_t TSC_InitTypeDef::IODefaultMode***
  IO default mode This parameter can be a value of ***TSC_IO_Default_Mode***
- ***uint32_t TSC_InitTypeDef::SynchroPinPolarity***
  Synchro pin polarity This parameter can be a value of ***TSC_Synchro_Pin_Polarity***
- ***uint32_t TSC_InitTypeDef::AcquisitionMode***
  Acquisition mode This parameter can be a value of ***TSC_Acquisition_Mode***
- ***uint32_t TSC_InitTypeDef::MaxCountInterrupt***
  Max count interrupt activation This parameter can be set to ENABLE or DISABLE.

- ***uint32_t TSC_InitTypeDef::ChannelIOs***
  Channel IOs mask
- ***uint32_t TSC_InitTypeDef::ShieldIOs***
  Shield IOs mask
- ***uint32_t TSC_InitTypeDef::SamplingIOs***
  Sampling IOs mask

67.1.2 TSC_IOConfigTypeDef

Data Fields

- ***uint32_t ChannelIOs***
- ***uint32_t ShieldIOs***
- ***uint32_t SamplingIOs***

Field Documentation

- ***uint32_t TSC_IOConfigTypeDef::ChannelIOs***
  Channel IOs mask
- ***uint32_t TSC_IOConfigTypeDef::ShieldIOs***
  Shield IOs mask
- ***uint32_t TSC_IOConfigTypeDef::SamplingIOs***
  Sampling IOs mask

67.1.3 TSC_HandleTypeDef

Data Fields

- ***TSC_TypeDef * Instance***
- ***TSC_InitTypeDef Init***
- ***__IO HAL_TSC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***

Field Documentation

- ***TSC_TypeDef* TSC_HandleTypeDef::Instance***
  Register base address
- ***TSC_InitTypeDef TSC_HandleTypeDef::Init***
  Initialization parameters
- ***__IO HAL_TSC_StateTypeDef TSC_HandleTypeDef::State***
  Peripheral state
- ***HAL_LockTypeDef TSC_HandleTypeDef::Lock***
  Lock feature

67.2 TSC Firmware driver API description

67.2.1 TSC specific features

1. Proven and robust surface charge transfer acquisition principle
2. Supports up to 3 capacitive sensing channels per group
3. Capacitive sensing channels can be acquired in parallel offering a very good response time
4. Spread spectrum feature to improve system robustness in noisy environments
5. Full hardware management of the charge transfer acquisition sequence
6. Programmable charge transfer frequency
7. Programmable sampling capacitor I/O pin
8. Programmable channel I/O pin
9. Programmable max count value to avoid long acquisition when a channel is faulty

10. Dedicated end of acquisition and max count error flags with interrupt capability
11. One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
12. Compatible with proximity, touchkey, linear and rotary touch sensor implementation

67.2.2 How to use this driver

1. Enable the TSC interface clock using __HAL_RCC_TSC_CLK_ENABLE() macro.
2. GPIO pins configuration
   - Enable the clock for the TSC GPIOs using __HAL_RCC_GPIOx_CLK_ENABLE() macro.
   - Configure the TSC pins used as sampling IOs in alternate function output Open-Drain mode, and TSC pins used as channel/shield IOs in alternate function output Push-Pull mode using HAL_GPIO_Init() function.
3. Interrupts configuration
   - Configure the NVIC (if the interrupt model is used) using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ() and function.
4. TSC configuration
   - Configure all TSC parameters and used TSC IOs using HAL_TSC_Init() function.

TSC peripheral alternate functions are mapped on AF9.

Acquisition sequence

- Discharge all IOs using HAL_TSC_IODischarge() function.
- Wait a certain time allowing a good discharge of all capacitors. This delay depends of the sampling capacitor and electrodes design.
- Select the channel IOs to be acquired using HAL_TSC_IOConfig() function.
- Launch the acquisition using either HAL_TSC_Start() or HAL_TSC_Start_IT() function. If the synchronized mode is selected, the acquisition will start as soon as the signal is received on the synchro pin.
- Wait the end of acquisition using either HAL_TSC_PollForAcquisition() or HAL_TSC_GetState() function or using WFI instruction for example.
- Check the group acquisition status using HAL_TSC_GroupGetStatus() function.
- Read the acquisition value using HAL_TSC_GroupGetValue() function.

67.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the TSC.
- De-initialize the TSC.

This section contains the following APIs:

- ***HAL_TSC_Init()***
- ***HAL_TSC_DeInit()***
- ***HAL_TSC_MspInit()***
- ***HAL_TSC_MspDeInit()***

67.2.4 IO Operation functions

This section provides functions allowing to:

- Start acquisition in polling mode.
- Start acquisition in interrupt mode.
- Stop conversion in polling mode.
- Stop conversion in interrupt mode.

- Poll for acquisition completed.
- Get group acquisition status.
- Get group acquisition value.

This section contains the following APIs:

- *HAL_TSC_Start()*
- *HAL_TSC_Start_IT()*
- *HAL_TSC_Stop()*
- *HAL_TSC_Stop_IT()*
- *HAL_TSC_PollForAcquisition()*
- *HAL_TSC_GroupGetStatus()*
- *HAL_TSC_GroupGetValue()*

67.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure TSC IOs
- Discharge TSC IOs

This section contains the following APIs:

- *HAL_TSC_IOConfig()*
- *HAL_TSC_IODischarge()*

67.2.6 State and Errors functions

This subsection provides functions allowing to

- Get TSC state.

This section contains the following APIs:

- *HAL_TSC_GetState()*

67.2.7 Detailed description of functions

HAL_TSC_Init

Function name	**HAL_StatusTypeDef HAL_TSC_Init (TSC_HandleTypeDef * htsc)**
Function description	Initialize the TSC peripheral according to the specified parameters in the TSC_InitTypeDef structure and initialize the associated handle.
Parameters	• **htsc:** TSC handle
Return values	• **HAL:** status

HAL_TSC_DeInit

Function name	**HAL_StatusTypeDef HAL_TSC_DeInit (TSC_HandleTypeDef * htsc)**
Function description	Deinitialize the TSC peripheral registers to their default reset values.
Parameters	• **htsc:** TSC handle

Return values	• **HAL:** status

HAL_TSC_MspInit

Function name	**void HAL_TSC_MspInit (TSC_HandleTypeDef * htsc)**
Function description	Initialize the TSC MSP.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **None:**

HAL_TSC_MspDeInit

Function name	**void HAL_TSC_MspDeInit (TSC_HandleTypeDef * htsc)**
Function description	DeInitialize the TSC MSP.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **None:**

HAL_TSC_Start

Function name	**HAL_StatusTypeDef HAL_TSC_Start (TSC_HandleTypeDef * htsc)**
Function description	Start the acquisition.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** status

HAL_TSC_Start_IT

Function name	**HAL_StatusTypeDef HAL_TSC_Start_IT (TSC_HandleTypeDef * htsc)**
Function description	Start the acquisition in interrupt mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** status.

HAL_TSC_Stop

Function name	**HAL_StatusTypeDef HAL_TSC_Stop (TSC_HandleTypeDef * htsc)**
Function description	Stop the acquisition previously launched in polling mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** status

HAL_TSC_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TSC_Stop_IT (TSC_HandleTypeDef * htsc)**
Function description	Stop the acquisition previously launched in interrupt mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** status

HAL_TSC_PollForAcquisition

Function name	**HAL_StatusTypeDef HAL_TSC_PollForAcquisition (TSC_HandleTypeDef * htsc)**
Function description	Start acquisition and wait until completion.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** state
Notes	• There is no need of a timeout parameter as the max count error is already managed by the TSC peripheral.

HAL_TSC_GroupGetStatus

Function name	**TSC_GroupStatusTypeDef HAL_TSC_GroupGetStatus (TSC_HandleTypeDef * htsc, uint32_t gx_index)**
Function description	Get the acquisition status for a group.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. • **gx_index:** Index of the group
Return values	• **Group:** status

HAL_TSC_GroupGetValue

Function name	**uint32_t HAL_TSC_GroupGetValue (TSC_HandleTypeDef * htsc, uint32_t gx_index)**
Function description	Get the acquisition measure for a group.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. • **gx_index:** Index of the group
Return values	• **Acquisition:** measure

HAL_TSC_IOConfig

Function name	**HAL_StatusTypeDef HAL_TSC_IOConfig (TSC_HandleTypeDef * htsc, TSC_IOConfigTypeDef * config)**
Function description	Configure TSC IOs.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

- **config:** pointer to the configuration structure.

Return values
- **HAL:** status

HAL_TSC_IODischarge

Function name	**HAL_StatusTypeDef HAL_TSC_IODischarge (TSC_HandleTypeDef * htsc, uint32_t choice)**
Function description	Discharge TSC IOs.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. • **choice:** enable or disable
Return values	• **HAL:** status

HAL_TSC_GetState

Function name	**HAL_TSC_StateTypeDef HAL_TSC_GetState (TSC_HandleTypeDef * htsc)**
Function description	Return the TSC handle state.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **HAL:** state

HAL_TSC_IRQHandler

Function name	**void HAL_TSC_IRQHandler (TSC_HandleTypeDef * htsc)**
Function description	Handle TSC interrupt request.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **None:**

HAL_TSC_ConvCpltCallback

Function name	**void HAL_TSC_ConvCpltCallback (TSC_HandleTypeDef * htsc)**
Function description	Acquisition completed callback in non-blocking mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• **None:**

HAL_TSC_ErrorCallback

Function name	**void HAL_TSC_ErrorCallback (TSC_HandleTypeDef * htsc)**
Function description	Error callback in non-blocking mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values • **None:**

67.3 TSC Firmware driver defines

67.3.1 TSC

Acquisition Mode

TSC_ACQ_MODE_NORMAL

TSC_ACQ_MODE_SYNCHRO

CTPulse High Length

TSC_CTPH_1CYCLE

TSC_CTPH_2CYCLES

TSC_CTPH_3CYCLES

TSC_CTPH_4CYCLES

TSC_CTPH_5CYCLES

TSC_CTPH_6CYCLES

TSC_CTPH_7CYCLES

TSC_CTPH_8CYCLES

TSC_CTPH_9CYCLES

TSC_CTPH_10CYCLES

TSC_CTPH_11CYCLES

TSC_CTPH_12CYCLES

TSC_CTPH_13CYCLES

TSC_CTPH_14CYCLES

TSC_CTPH_15CYCLES

TSC_CTPH_16CYCLES

CTPulse Low Length

TSC_CTPL_1CYCLE

TSC_CTPL_2CYCLES

TSC_CTPL_3CYCLES

TSC_CTPL_4CYCLES

TSC_CTPL_5CYCLES

TSC_CTPL_6CYCLES

TSC_CTPL_7CYCLES

TSC_CTPL_8CYCLES

TSC_CTPL_9CYCLES

TSC_CTPL_10CYCLES

TSC_CTPL_11CYCLES

TSC_CTPL_12CYCLES

TSC_CTPL_13CYCLES

TSC_CTPL_14CYCLES

TSC_CTPL_15CYCLES

TSC_CTPL_16CYCLES

TSC Exported Macros

__HAL_TSC_RESET_HANDLE_STATE

Description:

- Reset TSC handle state.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_ENABLE

Description:

- Enable the TSC peripheral.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_DISABLE

Description:

- Disable the TSC peripheral.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_START_ACQ

Description:

- Start acquisition.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_STOP_ACQ

Description:

- Stop acquisition.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_SET_IODEF_OUTPPLOW	**Description:** • Set IO default mode to output push-pull low. **Parameters:** • __HANDLE__: TSC handle **Return value:** • None
__HAL_TSC_SET_IODEF_INFLOAT	**Description:** • Set IO default mode to input floating. **Parameters:** • __HANDLE__: TSC handle **Return value:** • None
__HAL_TSC_SET_SYNC_POL_FALL	**Description:** • Set synchronization polarity to falling edge. **Parameters:** • __HANDLE__: TSC handle **Return value:** • None
__HAL_TSC_SET_SYNC_POL_RISE_HIGH	**Description:** • Set synchronization polarity to rising edge and high level. **Parameters:** • __HANDLE__: TSC handle **Return value:** • None
__HAL_TSC_ENABLE_IT	**Description:** • Enable TSC interrupt. **Parameters:** • __HANDLE__: TSC handle • __INTERRUPT__: TSC interrupt **Return value:** • None
__HAL_TSC_DISABLE_IT	**Description:** • Disable TSC interrupt. **Parameters:**

- __HANDLE__: TSC handle
- __INTERRUPT__: TSC interrupt

Return value:

- None

__HAL_TSC_GET_IT_SOURCE

Description:

- Check whether the specified TSC interrupt source is enabled or not.

Parameters:

- __HANDLE__: TSC Handle
- __INTERRUPT__: TSC interrupt

Return value:

- SET: or RESET

__HAL_TSC_GET_FLAG

Description:

- Check whether the specified TSC flag is set or not.

Parameters:

- __HANDLE__: TSC handle
- __FLAG__: TSC flag

Return value:

- SET: or RESET

__HAL_TSC_CLEAR_FLAG

Description:

- Clear the TSC's pending flag.

Parameters:

- __HANDLE__: TSC handle
- __FLAG__: TSC flag

Return value:

- None

__HAL_TSC_ENABLE_HYSTERESIS

Description:

- Enable schmitt trigger hysteresis on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_DISABLE_HYSTERESIS

Description:

- Disable schmitt trigger hysteresis on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_OPEN_ANALOG_SWITCH

Description:

- Open analog switch on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_CLOSE_ANALOG_SWITCH

Description:

- Close analog switch on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_ENABLE_CHANNEL

Description:

- Enable a group of IOs in channel mode.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_DISABLE_CHANNEL

Description:

- Disable a group of channel IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_ENABLE_SAMPLING

Description:

- Enable a group of IOs in sampling mode.

Parameters:

- __HANDLE__: TSC handle

- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_DISABLE_SAMPLING

Description:

- Disable a group of sampling IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_ENABLE_GROUP

Description:

- Enable acquisition groups.

Parameters:

- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:

- None

__HAL_TSC_DISABLE_GROUP

Description:

- Disable acquisition groups.

Parameters:

- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:

- None

__HAL_TSC_GET_GROUP_STATUS

Description:

- Gets acquisition group status.

Parameters:

- __HANDLE__: TSC Handle
- __GX_INDEX__: Group index

Return value:

- SET: or RESET

Flags definition

TSC_FLAG_EOA

TSC_FLAG_MCE

Group definition

TSC_NB_OF_GROUPS

TSC_GROUP1

TSC_GROUP2

TSC_GROUP3

TSC_GROUP4

TSC_GROUP5

TSC_GROUP6

TSC_GROUP7

TSC_GROUP8

TSC_ALL_GROUPS

TSC_GROUP1_IDX

TSC_GROUP2_IDX

TSC_GROUP3_IDX

TSC_GROUP4_IDX

TSC_GROUP5_IDX

TSC_GROUP6_IDX

TSC_GROUP7_IDX

TSC_GROUP8_IDX

TSC_GROUP1_IO1

TSC_GROUP1_IO2

TSC_GROUP1_IO3

TSC_GROUP1_IO4

TSC_GROUP1_ALL_IOS

TSC_GROUP2_IO1

TSC_GROUP2_IO2

TSC_GROUP2_IO3

TSC_GROUP2_IO4

TSC_GROUP2_ALL_IOS

TSC_GROUP3_IO1

TSC_GROUP3_IO2

TSC_GROUP3_IO3

TSC_GROUP3_IO4

TSC_GROUP3_ALL_IOS

TSC_GROUP4_IO1

TSC_GROUP4_IO2

TSC_GROUP4_IO3

TSC_GROUP4_IO4

TSC_GROUP4_ALL_IOS

TSC_GROUP5_IO1

TSC_GROUP5_IO2

TSC_GROUP5_IO3

TSC_GROUP5_IO4

TSC_GROUP5_ALL_IOS

TSC_GROUP6_IO1

TSC_GROUP6_IO2

TSC_GROUP6_IO3

TSC_GROUP6_IO4

TSC_GROUP6_ALL_IOS

TSC_GROUP7_IO1

TSC_GROUP7_IO2

TSC_GROUP7_IO3

TSC_GROUP7_IO4

TSC_GROUP7_ALL_IOS

TSC_GROUP8_IO1

TSC_GROUP8_IO2

TSC_GROUP8_IO3

TSC_GROUP8_IO4

TSC_GROUP8_ALL_IOS

TSC_ALL_GROUPS_ALL_IOS

Interrupts definition

TSC_IT_EOA

TSC_IT_MCE

IO Default Mode

TSC_IODEF_OUT_PP_LOW

TSC_IODEF_IN_FLOAT

IO Mode

TSC_IOMODE_UNUSED

TSC_IOMODE_CHANNEL

TSC_IOMODE_SHIELD

TSC_IOMODE_SAMPLING

Max Count Value

TSC_MCV_255

TSC_MCV_511

TSC_MCV_1023

TSC_MCV_2047

TSC_MCV_4095

TSC_MCV_8191

TSC_MCV_16383

Pulse Generator Prescaler

TSC_PG_PRESC_DIV1

TSC_PG_PRESC_DIV2

TSC_PG_PRESC_DIV4

TSC_PG_PRESC_DIV8

TSC_PG_PRESC_DIV16

TSC_PG_PRESC_DIV32

TSC_PG_PRESC_DIV64

TSC_PG_PRESC_DIV128

Spread Spectrum Prescaler

TSC_SS_PRESC_DIV1

TSC_SS_PRESC_DIV2

Synchro Pin Polarity

TSC_SYNC_POLARITY_FALLING

TSC_SYNC_POLARITY_RISING

68 HAL UART Generic Driver

68.1 UART Firmware driver registers structures

68.1.1 UART_InitTypeDef

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t HwFlowCtl***
- ***uint32_t OverSampling***
- ***uint32_t OneBitSampling***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***
  This member configures the UART communication baud rate. The baud rate register is computed using the following formula: UART: =====If oversampling is 16 or in LIN mode, Baud Rate Register = ((uart_ker_ckpres) / ((huart->Init.BaudRate)))If oversampling is 8, Baud Rate Register[15:4] = ((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[3:0]) >> 1 LPUART: ======= Baud Rate Register = ((256 * lpuart_ker_ckpres) / ((huart->Init.BaudRate))) where (uart/lpuart)_ker_ck_pres is the UART input clock divided by a prescaler
- ***uint32_t UART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***UARTEx_Word_Length***.
- ***uint32_t UART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***.
- ***uint32_t UART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***UART_Parity***
  Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t UART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***.
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***.
- ***uint32_t UART_InitTypeDef::OverSampling***
  Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to f_PCLK/8). This parameter can be a value of ***UART_Over_Sampling***.
- ***uint32_t UART_InitTypeDef::OneBitSampling***
  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***UART_OneBit_Sampling***.

- ***uint32_t UART_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the UART clock source. This parameter can be a value of ***UART_ClockPrescaler***.

68.1.2 UART_AdvFeatureInitTypeDef

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t AutoBaudRateEnable***
- ***uint32_t AutoBaudRateMode***
- ***uint32_t MSBFirst***

Field Documentation

- ***uint32_t UART_AdvFeatureInitTypeDef::AdvFeatureInit***
  Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of ***UART_Advanced_Features_Initialization_Type***.
- ***uint32_t UART_AdvFeatureInitTypeDef::TxPinLevelInvert***
  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***UART_Tx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***UART_Rx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***UART_Data_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***UART_Rx_Tx_Swap***.
- ***uint32_t UART_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***UART_Overrun_Disable***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***UART_DMA_Disable_on_Rx_Error***.
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateEnable***
  Specifies whether auto Baud rate detection is enabled. This parameter can be a value of ***UART_AutoBaudRate_Enable***
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateMode***
  If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of ***UART_AutoBaud_Rate_Mode***.
- ***uint32_t UART_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***UART_MSB_First***.

68.1.3 __UART_HandleTypeDef

Data Fields

- ***USART_TypeDef * Instance***
- ***UART_InitTypeDef Init***
- ***UART_AdvFeatureInitTypeDef AdvancedInit***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t FifoMode***
- ***uint32_t SlaveMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_UART_StateTypeDef gState***
- ***__IO HAL_UART_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* __UART_HandleTypeDef::Instance***
  UART registers base address
- ***UART_InitTypeDef __UART_HandleTypeDef::Init***
  UART communication parameters
- ***UART_AdvFeatureInitTypeDef __UART_HandleTypeDef::AdvancedInit***
  UART Advanced Features initialization parameters
- ***uint8_t* __UART_HandleTypeDef::pTxBuffPtr***
  Pointer to UART Tx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::TxXferSize***
  UART Tx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::TxXferCount***
  UART Tx Transfer Counter
- ***uint8_t* __UART_HandleTypeDef::pRxBuffPtr***
  Pointer to UART Rx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::RxXferSize***
  UART Rx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::RxXferCount***
  UART Rx Transfer Counter
- ***uint16_t __UART_HandleTypeDef::Mask***
  UART Rx RDR register mask
- ***uint16_t __UART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __UART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __UART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode is being used. This parameter can be a value of ***UARTEx_FIFO_mode***.

- ***uint32_t __UART_HandleTypeDef::SlaveMode***
  Specifies if the UART SPI Slave mode is being used. This parameter can be a value of ***UARTEx_Slave_Mode***.
- ***void(* __UART_HandleTypeDef::RxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __UART_HandleTypeDef::TxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmatx***
  UART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmarx***
  UART Rx DMA Handle parameters
- ***HAL_LockTypeDef __UART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::gState***
  UART state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::RxState***
  UART state information related to Rx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO uint32_t __UART_HandleTypeDef::ErrorCode***
  UART Error code

68.2 UART Firmware driver API description

68.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a UART_HandleTypeDef handle structure (eg. UART_HandleTypeDef huart).
2. Initialize the UART low level resources by implementing the HAL_UART_MspInit() API:
   - Enable the USARTx interface clock.
   - UART pins configuration:
     - Enable the clock for the UART GPIOs.
     - Configure these UART pins as alternate function pull-up.
   - NVIC configuration if you need to use interrupt process (HAL_UART_Transmit_IT() and HAL_UART_Receive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - UART interrupts handling: The specific UART interrupts (Transmission complete interrupt, RXNE interrupt, RX/TX FIFOs related interrupts and Error Interrupts) are managed using the macros __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive processes.
   - DMA Configuration if you need to use DMA process (HAL_UART_Transmit_DMA() and HAL_UART_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.

3. Program the Baud Rate, Word Length, Stop Bit, Parity, Prescaler value , Hardware flow control and Mode (Receiver/Transmitter) in the huart handle Init structure.
4. If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...) in the huart handle AdvancedInit structure.
5. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
6. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
7. For the UART LIN (Local Interconnection Network) mode, initialize the UART registers by calling the HAL_LIN_Init() API.
8. For the UART Multiprocessor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.
9. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.

These API's (HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init(), also configure the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

68.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init()and HAL_MultiProcessor_Init()API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and UART multiprocessor mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_UART_Init()***
- ***HAL_HalfDuplex_Init()***
- ***HAL_LIN_Init()***
- ***HAL_MultiProcessor_Init()***

- *HAL_UART_DeInit()*
- *HAL_UART_MspInit()*
- *HAL_UART_MspDeInit()*

68.2.3 IO operation functions

This section contains the following APIs:

- *HAL_UART_Transmit()*
- *HAL_UART_Receive()*
- *HAL_UART_Transmit_IT()*
- *HAL_UART_Receive_IT()*
- *HAL_UART_Transmit_DMA()*
- *HAL_UART_Receive_DMA()*
- *HAL_UART_DMAPause()*
- *HAL_UART_DMAResume()*
- *HAL_UART_DMAStop()*
- *HAL_UART_Abort()*
- *HAL_UART_AbortTransmit()*
- *HAL_UART_AbortReceive()*
- *HAL_UART_Abort_IT()*
- *HAL_UART_AbortTransmit_IT()*
- *HAL_UART_AbortReceive_IT()*
- *HAL_UART_IRQHandler()*
- *HAL_UART_TxCpltCallback()*
- *HAL_UART_TxHalfCpltCallback()*
- *HAL_UART_RxCpltCallback()*
- *HAL_UART_RxHalfCpltCallback()*
- *HAL_UART_ErrorCallback()*
- *HAL_UART_AbortCpltCallback()*
- *HAL_UART_AbortTransmitCpltCallback()*
- *HAL_UART_AbortReceiveCpltCallback()*

68.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART.

- HAL_MultiProcessor_EnableMuteMode() API enables mute mode
- HAL_MultiProcessor_DisableMuteMode() API disables mute mode
- HAL_MultiProcessor_EnterMuteMode() API enters mute mode
- HAL_MultiProcessor_EnableMuteMode() API enables mute mode
- UART_SetConfig() API configures the UART peripheral
- UART_AdvFeatureConfig() API optionally configures the UART advanced features
- UART_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization
- UART_Wakeup_AddressConfig() API configures the wake-up from stop mode parameters
- HAL_HalfDuplex_EnableTransmitter() API disables receiver and enables transmitter
- HAL_HalfDuplex_EnableReceiver() API disables transmitter and enables receiver
- HAL_LIN_SendBreak() API transmits the break characters

This section contains the following APIs:

- *HAL_MultiProcessor_EnableMuteMode()*
- *HAL_MultiProcessor_DisableMuteMode()*
- *HAL_MultiProcessor_EnterMuteMode()*

- *HAL_HalfDuplex_EnableTransmitter()*
- *HAL_HalfDuplex_EnableReceiver()*
- *HAL_LIN_SendBreak()*

68.2.5 Peripheral State and Error functions

This subsection provides functions allowing to:

- Return the UART handle state.
- Return the UART handle error code

This section contains the following APIs:

- *HAL_UART_GetState()*
- *HAL_UART_GetError()*

68.2.6 Detailed description of functions

HAL_UART_Init

Function name	**HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)**
Function description	Initialize the UART mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_HalfDuplex_Init

Function name	**HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)**
Function description	Initialize the half-duplex mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_LIN_Init

Function name	**HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)**
Function description	Initialize the LIN mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle .
Parameters	• **huart:** UART handle. • **BreakDetectLength:** Specifies the LIN break detection length. This parameter can be one of the following values: – UART_LINBREAKDETECTLENGTH_10B 10-bit break detection – UART_LINBREAKDETECTLENGTH_11B 11-bit break detection
Return values	• **HAL:** status

HAL_MultiProcessor_Init

Function name	**HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)**
Function description	Initialize the multiprocessor mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.
Parameters	• **huart:** UART handle. • **Address:** UART node address (4-, 6-, 7- or 8-bit long). • **WakeUpMethod:** Specifies the UART wakeup method. This parameter can be one of the following values: – UART_WAKEUPMETHOD_IDLELINE WakeUp by an idle line detection – UART_WAKEUPMETHOD_ADDRESSMARK WakeUp by an address mark
Return values	• **HAL:** status
Notes	• If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function. • If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API HAL_MultiProcessorEx_AddressLength_Set() must be called after HAL_MultiProcessor_Init().

HAL_UART_DeInit

Function name	**HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)**
Function description	DeInitialize the UART peripheral.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_UART_MspInit

Function name	**void HAL_UART_MspInit (UART_HandleTypeDef * huart)**
Function description	Initialize the UART MSP.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_MspDeInit

Function name	**void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)**
Function description	DeInitialize the UART MSP.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_Transmit

Function name	**HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Send an amount of data in blocking mode.
Parameters	• **huart:** UART handle. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be sent. • **Timeout:** Timeout duration.
Return values	• **HAL:** status
Notes	• When FIFO mode is enabled, writing a data in the TDR register adds one data to the TXFIFO. Write operations to the TDR register are performed when TXFNF flag is set. From hardware perspective, TXFNF flag and TXE are mapped on the same bit-field.

HAL_UART_Receive

Function name	**HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **huart:** UART handle. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be received. • **Timeout:** Timeout duration.
Return values	• **HAL:** status
Notes	• When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_UART_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function description	Send an amount of data in interrupt mode.
Parameters	• **huart:** UART handle. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be sent.
Return values	• **HAL:** status

HAL_UART_Receive_IT

Function name	**HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**

Function description	Receive an amount of data in interrupt mode.
Parameters	• **huart:** UART handle. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be received.
Return values	• **HAL:** status

HAL_UART_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function description	Send an amount of data in DMA mode.
Parameters	• **huart:** UART handle. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be sent.
Return values	• **HAL:** status

HAL_UART_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function description	Receive an amount of data in DMA mode.
Parameters	• **huart:** UART handle. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be received.
Return values	• **HAL:** status
Notes	• When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).

HAL_UART_DMAPause

Function name	**HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)**
Function description	Pause the DMA Transfer.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_UART_DMAResume

Function name	**HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)**
Function description	Resume the DMA Transfer.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_UART_DMAStop

Function name	**HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)**
Function description	Stop the DMA Transfer.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_UART_Abort

Function name	**HAL_StatusTypeDef HAL_UART_Abort (UART_HandleTypeDef * huart)**
Function description	Abort ongoing transfers (blocking mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode: when exiting function, Abort is considered as completed.

HAL_UART_AbortTransmit

Function name	**HAL_StatusTypeDef HAL_UART_AbortTransmit (UART_HandleTypeDef * huart)**
Function description	Abort ongoing Transmit transfer (blocking mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode: when exiting function, Abort is considered as completed.

HAL_UART_AbortReceive

Function name	**HAL_StatusTypeDef HAL_UART_AbortReceive (UART_HandleTypeDef * huart)**
Function description	Abort ongoing Receive transfer (blocking mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode: when exiting function, Abort is considered as completed.

HAL_UART_Abort_IT

Function name	**HAL_StatusTypeDef HAL_UART_Abort_IT (UART_HandleTypeDef * huart)**
Function description	Abort ongoing transfers (Interrupt mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortTransmit_IT

Function name	**HAL_StatusTypeDef HAL_UART_AbortTransmit_IT (UART_HandleTypeDef * huart)**
Function description	Abort ongoing Transmit transfer (Interrupt mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortReceive_IT

Function name	**HAL_StatusTypeDef HAL_UART_AbortReceive_IT**

	(UART_HandleTypeDef * huart)
Function description	Abort ongoing Receive transfer (Interrupt mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_IRQHandler

Function name	**void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)**
Function description	Handle UART interrupt request.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_TxHalfCpltCallback

Function name	**void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)**
Function description	Tx Half Transfer completed callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_TxCpltCallback

Function name	**void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)**
Function description	Tx Transfer completed callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_RxHalfCpltCallback

Function name	**void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)**
Function description	Rx Half Transfer completed callback.
Parameters	• **huart:** UART handle.

Return values	• **None:**

HAL_UART_RxCpltCallback

Function name	**void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)**
Function description	Rx Transfer completed callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_ErrorCallback

Function name	**void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)**
Function description	UART error callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_AbortCpltCallback

Function name	**void HAL_UART_AbortCpltCallback (UART_HandleTypeDef * huart)**
Function description	UART Abort Complete callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_AbortTransmitCpltCallback

Function name	**void HAL_UART_AbortTransmitCpltCallback (UART_HandleTypeDef * huart)**
Function description	UART Abort Complete callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_AbortReceiveCpltCallback

Function name	**void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef * huart)**
Function description	UART Abort Receive Complete callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_LIN_SendBreak

Function name	**HAL_StatusTypeDef HAL_LIN_SendBreak**

	(UART_HandleTypeDef * huart)
Function description	Transmit break characters.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_MultiProcessor_EnableMuteMode

Function name	**HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)**
Function description	Enable UART in mute mode (does not mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_MultiProcessor_DisableMuteMode

Function name	**HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)**
Function description	Disable UART mute mode (does not mean the UART actually exits mute mode as it may not have been in mute mode at this very moment).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_MultiProcessor_EnterMuteMode

Function name	**void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)**
Function description	Enter UART mute mode (means UART actually enters mute mode).
Parameters	• **huart:** UART handle.
Return values	• **None:**
Notes	• To exit from mute mode, HAL_MultiProcessor_DisableMuteMode() API must be called.

HAL_HalfDuplex_EnableTransmitter

Function name	**HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)**
Function description	Enable the UART transmitter and disable the UART receiver.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_HalfDuplex_EnableReceiver

Function name	**HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)**
Function description	Enable the UART receiver and disable the UART transmitter.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status.

HAL_UART_GetState

Function name	**HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)**
Function description	Return the UART handle state.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	• **HAL:** state

HAL_UART_GetError

Function name	**uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)**
Function description	Return the UART handle error code.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	• **UART:** Error Code

UART_SetConfig

Function name	**HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)**
Function description	Configure the UART peripheral.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

UART_CheckIdleState

Function name	**HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)**
Function description	Check the UART Idle State.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

UART_WaitOnFlagUntilTimeout

Function name	**HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Tickstart, uint32_t Timeout)**

Function description	Handle UART Communication Timeout.
Parameters	• **huart:** UART handle. • **Flag:** Specifies the UART flag to check • **Status:** Flag status (SET or RESET) • **Tickstart:** Tick start value • **Timeout:** Timeout duration
Return values	• **HAL:** status

UART_AdvFeatureConfig

Function name	**void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)**
Function description	Configure the UART peripheral advanced features.
Parameters	• **huart:** UART handle.
Return values	• **None:**

68.3 UART Firmware driver defines

68.3.1 UART

UART Advanced Feature Initialization Type

UART_ADVFEATURE_NO_INIT	No advanced feature initialization
UART_ADVFEATURE_TXINVERT_INIT	TX pin active level inversion
UART_ADVFEATURE_RXINVERT_INIT	RX pin active level inversion
UART_ADVFEATURE_DATAINVERT_INIT	Binary data inversion
UART_ADVFEATURE_SWAP_INIT	TX/RX pins swap
UART_ADVFEATURE_RXOVERRUNDISABLE_INIT	RX overrun disable
UART_ADVFEATURE_DMADISABLEONERROR_INIT	DMA disable on Reception Error
UART_ADVFEATURE_AUTOBAUDRATE_INIT	Auto Baud rate detection initialization
UART_ADVFEATURE_MSBFIRST_INIT	Most significant bit sent/received first

UART Advanced Feature Auto BaudRate Enable

UART_ADVFEATURE_AUTOBAUDRATE_DISABLE	RX Auto Baud rate detection enable
UART_ADVFEATURE_AUTOBAUDRATE_ENABLE	RX Auto Baud rate detection disable

UART Advanced Feature AutoBaud Rate Mode

UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT	Auto Baud rate detection on start bit
UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE	Auto Baud rate detection on falling edge
UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFRAME	Auto Baud rate detection on 0x7F frame detection

UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME	Auto Baud rate detection on 0x55 frame detection

UART Clock Prescaler

UART_PRESCALER_DIV1	fclk_pres = fclk
UART_PRESCALER_DIV2	fclk_pres = fclk/2
UART_PRESCALER_DIV4	fclk_pres = fclk/4
UART_PRESCALER_DIV6	fclk_pres = fclk/6
UART_PRESCALER_DIV8	fclk_pres = fclk/8
UART_PRESCALER_DIV10	fclk_pres = fclk/10
UART_PRESCALER_DIV12	fclk_pres = fclk/12
UART_PRESCALER_DIV16	fclk_pres = fclk/16
UART_PRESCALER_DIV32	fclk_pres = fclk/32
UART_PRESCALER_DIV64	fclk_pres = fclk/64
UART_PRESCALER_DIV128	fclk_pres = fclk/128
UART_PRESCALER_DIV256	fclk_pres = fclk/256

UART Driver Enable Assertion Time LSB Position In CR1 Register

UART_CR1_DEAT_ADDRESS_LSB_POS	UART Driver Enable assertion time LSB position in CR1 register

UART Driver Enable DeAssertion Time LSB Position In CR1 Register

UART_CR1_DEDT_ADDRESS_LSB_POS	UART Driver Enable de-assertion time LSB position in CR1 register

UART Address-matching LSB Position In CR2 Register

UART_CR2_ADDRESS_LSB_POS	UART address-matching LSB position in CR2 register

UART Advanced Feature Binary Data Inversion

UART_ADVFEATURE_DATAINV_DISABLE	Binary data inversion disable
UART_ADVFEATURE_DATAINV_ENABLE	Binary data inversion enable

UART Advanced Feature DMA Disable On Rx Error

UART_ADVFEATURE_DMA_ENABLEONRXERROR	DMA enable on Reception Error
UART_ADVFEATURE_DMA_DISABLEONRXERROR	DMA disable on Reception Error

UART DMA Rx

UART_DMA_RX_DISABLE	UART DMA RX disabled
UART_DMA_RX_ENABLE	UART DMA RX enabled

UART DMA Tx

UART_DMA_TX_DISABLE	UART DMA TX disabled
UART_DMA_TX_ENABLE	UART DMA TX enabled

UART DriverEnable Polarity

UART_DE_POLARITY_HIGH	Driver enable signal is active high

UART_DE_POLARITY_LOW	Driver enable signal is active low

UART Exported Macros

__HAL_UART_RESET_HANDLE_STATE

Description:

- Reset UART handle states.

Parameters:

- __HANDLE__: UART handle.

Return value:

- None

__HAL_UART_FLUSH_DRREGISTER

Description:

- Flush the UART Data registers.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FLAG

Description:

- Clear the specified UART pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_TXFECF TXFIFO empty clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag

Return value:

- None

__HAL_UART_CLEAR_PEFLAG

Description:

- Clear the UART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FEFLAG

Description:

- Clear the UART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_NEFLAG

Description:

- Clear the UART NE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_OREFLAG

Description:

- Clear the UART ORE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_IDLEFLAG

Description:

- Clear the UART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_TXFECF

Description:

- Clear the UART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_GET_FLAG

Description:

- Check whether the specified UART flag is set or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - UART_FLAG_TXFT TXFIFO threshold flag
  - UART_FLAG_RXFT RXFIFO threshold flag
  - UART_FLAG_RXFF RXFIFO Full flag
  - UART_FLAG_TXFE TXFIFO Empty flag
  - UART_FLAG_REACK Receive enable acknowledge flag
  - UART_FLAG_TEACK Transmit enable acknowledge flag
  - UART_FLAG_WUF Wake up from stop mode flag
  - UART_FLAG_RWU Receiver wake up flag (if the UART in mute mode)
  - UART_FLAG_SBKF Send Break flag
  - UART_FLAG_CMF Character match flag
  - UART_FLAG_BUSY Busy flag
  - UART_FLAG_ABRF Auto Baud rate detection flag
  - UART_FLAG_ABRE Auto Baud rate detection error flag
  - UART_FLAG_CTS CTS Change flag
  - UART_FLAG_LBDF LIN Break detection flag
  - UART_FLAG_TXE Transmit data register empty flag
  - UART_FLAG_TXFNF UART TXFIFO not full flag
  - UART_FLAG_TC Transmission Complete flag
  - UART_FLAG_RXNE Receive data register not empty flag
  - UART_FLAG_RXFNE UART RXFIFO not empty flag
  - UART_FLAG_IDLE Idle Line detection flag
  - UART_FLAG_ORE Overrun Error flag
  - UART_FLAG_NE Noise Error flag
  - UART_FLAG_FE Framing Error flag
  - UART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_UART_ENABLE_IT

Description:

- Enable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_DISABLE_IT

Description:

- Disable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to disable. This parameter can be one of the following values:

- UART_IT_RXFF RXFIFO Full interrupt
- UART_IT_TXFE TXFIFO Empty interrupt
- UART_IT_RXFT RXFIFO threshold interrupt
- UART_IT_TXFT TXFIFO threshold interrupt
- UART_IT_WUF Wakeup from stop mode interrupt
- UART_IT_CM Character match interrupt
- UART_IT_CTS CTS change interrupt
- UART_IT_LBD LIN Break detection interrupt
- UART_IT_TXE Transmit Data Register empty interrupt
- UART_IT_TXFNF TX FIFO not full interrupt
- UART_IT_TC Transmission complete interrupt
- UART_IT_RXNE Receive Data register not empty interrupt
- UART_IT_RXFNE RXFIFO not empty interrupt
- UART_IT_IDLE Idle line detection interrupt
- UART_IT_PE Parity Error interrupt
- UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_GET_IT

Description:

- Check whether the specified UART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt

- UART_IT_LBD LIN Break detection interrupt
- UART_IT_TXE Transmit Data Register empty interrupt
- UART_IT_TXFNF TX FIFO not full interrupt
- UART_IT_TC Transmission complete interrupt
- UART_IT_RXNE Receive Data register not empty interrupt
- UART_IT_RXFNE RXFIFO not empty interrupt
- UART_IT_IDLE Idle line detection interrupt
- UART_IT_PE Parity Error interrupt
- UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_GET_IT_SOURCE

Description:

- Check whether the specified UART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty

interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_CLEAR_IT

Description:

- Clear the specified UART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_TXFECF TXFIFO empty Clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag

Return value:

- None

__HAL_UART_SEND_REQ

Description:

- Set a specific UART request flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following

values:
- UART_AUTOBAUD_REQUEST Auto-Baud Rate Request
- UART_SENDBREAK_REQUEST Send Break Request
- UART_MUTE_MODE_REQUEST Mute Mode Request
- UART_RXDATA_FLUSH_REQUEST Receive Data flush Request
- UART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:
- None

__HAL_UART_ONE_BIT_SAMPLE_ENABLE

Description:
- Enable the UART one bit sample method.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_ONE_BIT_SAMPLE_DISABLE

Description:
- Disable the UART one bit sample method.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_ENABLE

Description:
- Enable UART.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_DISABLE

Description:
- Disable UART.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_HWCONTROL_CTS_ENABLE

Description:
- Enable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled: UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_CTS_DISABLE

Description:

- Disable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled: UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_ENABLE

Description:

- Enable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled: UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_DISABLE

Description:

- Disable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled: UART instance should have already been initialised (through call of HAL_UART_Init()

)macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

UART Status Flags

UART_FLAG_TXFT	UART TXFIFO threshold flag
UART_FLAG_RXFT	UART RXFIFO threshold flag
UART_FLAG_RXFF	UART RXFIFO Full flag
UART_FLAG_TXFE	UART TXFIFO Empty flag
UART_FLAG_REACK	UART receive enable acknowledge flag
UART_FLAG_TEACK	UART transmit enable acknowledge flag
UART_FLAG_WUF	UART wake-up from stop mode flag
UART_FLAG_RWU	UART receiver wake-up from mute mode flag
UART_FLAG_SBKF	UART send break flag
UART_FLAG_CMF	UART character match flag
UART_FLAG_BUSY	UART busy flag
UART_FLAG_ABRF	UART auto Baud rate flag
UART_FLAG_ABRE	UART auto Baud rate error
UART_FLAG_CTS	UART clear to send flag
UART_FLAG_CTSIF	UART clear to send interrupt flag
UART_FLAG_LBDF	UART LIN break detection flag
UART_FLAG_TXE	UART transmit data register empty
UART_FLAG_TXFNF	UART TXFIFO not full
UART_FLAG_TC	UART transmission complete
UART_FLAG_RXNE	UART read data register not empty
UART_FLAG_RXFNE	UART RXFIFO not empty
UART_FLAG_IDLE	UART idle flag
UART_FLAG_ORE	UART overrun error
UART_FLAG_NE	UART noise error
UART_FLAG_FE	UART frame error
UART_FLAG_PE	UART parity error

UART Half Duplex Selection

UART_HALF_DUPLEX_DISABLE	UART half-duplex disabled
UART_HALF_DUPLEX_ENABLE	UART half-duplex enabled

UART Hardware Flow Control

UART_HWCONTROL_NONE	No hardware control
UART_HWCONTROL_RTS	Request To Send
UART_HWCONTROL_CTS	Clear To Send
UART_HWCONTROL_RTS_CTS	Request and Clear To Send

UART Interruptions Flag Mask

UART_IT_MASK	UART interruptions flags mask

UART Interrupts Definition

UART_IT_PE	UART parity error interruption
UART_IT_TXE	UART transmit data register empty interruption
UART_IT_TXFNF	UART TX FIFO not full interruption
UART_IT_TC	UART transmission complete interruption
UART_IT_RXNE	UART read data register not empty interruption
UART_IT_RXFNE	UART RXFIFO not empty interruption
UART_IT_IDLE	UART idle interruption
UART_IT_LBD	UART LIN break detection interruption
UART_IT_CTS	UART CTS interruption
UART_IT_CM	UART character match interruption
UART_IT_WUF	UART wake-up from stop mode interruption
UART_IT_RXFF	UART RXFIFO full interruption
UART_IT_TXFE	UART TXFIFO empty interruption
UART_IT_RXFT	UART RXFIFO threshold reached interruption
UART_IT_TXFT	UART TXFIFO threshold reached interruption
UART_IT_ERR	UART error interruption
UART_IT_ORE	UART overrun error interruption
UART_IT_NE	UART noise error interruption
UART_IT_FE	UART frame error interruption

UART Interruption Clear Flags

UART_CLEAR_PEF	Parity Error Clear Flag
UART_CLEAR_FEF	Framing Error Clear Flag
UART_CLEAR_NEF	Noise detected Clear Flag
UART_CLEAR_OREF	Overrun Error Clear Flag
UART_CLEAR_IDLEF	IDLE line detected Clear Flag
UART_CLEAR_TXFECF	TXFIFO empty clear flag
UART_CLEAR_TCF	Transmission Complete Clear Flag
UART_CLEAR_LBDF	LIN Break Detection Clear Flag

UART_CLEAR_CTSF	CTS Interrupt Clear Flag
UART_CLEAR_CMF	Character Match Clear Flag
UART_CLEAR_WUF	Wake Up from stop mode Clear Flag

UART Local Interconnection Network mode

UART_LIN_DISABLE	Local Interconnect Network disable
UART_LIN_ENABLE	Local Interconnect Network enable

UART LIN Break Detection

UART_LINBREAKDETECTLENGTH_10B	LIN 10-bit break detection length
UART_LINBREAKDETECTLENGTH_11B	LIN 11-bit break detection length

UART Transfer Mode

UART_MODE_RX	RX mode
UART_MODE_TX	TX mode
UART_MODE_TX_RX	RX and TX mode

UART Advanced Feature MSB First

UART_ADVFEATURE_MSBFIRST_DISABLE	Most significant bit sent/received first disable
UART_ADVFEATURE_MSBFIRST_ENABLE	Most significant bit sent/received first enable

UART Advanced Feature Mute Mode Enable

UART_ADVFEATURE_MUTEMODE_DISABLE	UART mute mode disable
UART_ADVFEATURE_MUTEMODE_ENABLE	UART mute mode enable

UART One Bit Sampling Method

UART_ONE_BIT_SAMPLE_DISABLE	One-bit sampling disable
UART_ONE_BIT_SAMPLE_ENABLE	One-bit sampling enable

UART Advanced Feature Overrun Disable

UART_ADVFEATURE_OVERRUN_ENABLE	RX overrun enable
UART_ADVFEATURE_OVERRUN_DISABLE	RX overrun disable

UART Over Sampling

UART_OVERSAMPLING_16	Oversampling by 16
UART_OVERSAMPLING_8	Oversampling by 8

UART Parity

UART_PARITY_NONE	No parity
UART_PARITY_EVEN	Even parity
UART_PARITY_ODD	Odd parity

UART Receiver TimeOut

UART_RECEIVER_TIMEOUT_DISABLE	UART receiver timeout disable
UART_RECEIVER_TIMEOUT_ENABLE	UART receiver timeout enable

UART Request Parameters

UART_AUTOBAUD_REQUEST	Auto-Baud Rate Request
UART_SENDBREAK_REQUEST	Send Break Request
UART_MUTE_MODE_REQUEST	Mute Mode Request
UART_RXDATA_FLUSH_REQUEST	Receive Data flush Request
UART_TXDATA_FLUSH_REQUEST	Transmit data flush Request

UART Advanced Feature RX Pin Active Level Inversion

UART_ADVFEATURE_RXINV_DISABLE	RX pin active level inversion disable
UART_ADVFEATURE_RXINV_ENABLE	RX pin active level inversion enable

UART Advanced Feature RX TX Pins Swap

UART_ADVFEATURE_SWAP_DISABLE	TX/RX pins swap disable
UART_ADVFEATURE_SWAP_ENABLE	TX/RX pins swap enable

UART State

UART_STATE_DISABLE	UART disabled
UART_STATE_ENABLE	UART enabled

UART Number of Stop Bits

UART_STOPBITS_0_5	UART frame with 0.5 stop bit
UART_STOPBITS_1	UART frame with 1 stop bit
UART_STOPBITS_1_5	UART frame with 1.5 stop bits
UART_STOPBITS_2	UART frame with 2 stop bits

UART Advanced Feature Stop Mode Enable

UART_ADVFEATURE_STOPMODE_DISABLE	UART stop mode disable
UART_ADVFEATURE_STOPMODE_ENABLE	UART stop mode enable

UART polling-based communications time-out value

HAL_UART_TIMEOUT_VALUE	UART polling-based communications time-out value

UART Advanced Feature TX Pin Active Level Inversion

UART_ADVFEATURE_TXINV_DISABLE	TX pin active level inversion disable
UART_ADVFEATURE_TXINV_ENABLE	TX pin active level inversion enable

UART WakeUp From Stop Selection

UART_WAKEUP_ON_ADDRESS	UART wake-up on address
UART_WAKEUP_ON_STARTBIT	UART wake-up on start bit
UART_WAKEUP_ON_READDATA_NONEMPTY	UART wake-up on receive data register not empty or RXFIFO is not empty

UART WakeUp Methods

UART_WAKEUPMETHOD_IDLELINE	UART wake-up on idle line
UART_WAKEUPMETHOD_ADDRESSMARK	UART wake-up on address mark

69 HAL UART Extension Driver

69.1 UARTEx Firmware driver registers structures

69.1.1 UART_WakeUpTypeDef

Data Fields

- ***uint32_t WakeUpEvent***
- ***uint16_t AddressLength***
- ***uint8_t Address***

Field Documentation

- ***uint32_t UART_WakeUpTypeDef::WakeUpEvent***
  Specifies which event will activat the Wakeup from Stop mode flag (WUF). This parameter can be a value of ***UART_WakeUp_from_Stop_Selection***. If set to UART_WAKEUP_ON_ADDRESS, the two other fields below must be filled up.
- ***uint16_t UART_WakeUpTypeDef::AddressLength***
  Specifies whether the address is 4 or 7-bit long. This parameter can be a value of ***UARTEx_WakeUp_Address_Length***.
- ***uint8_t UART_WakeUpTypeDef::Address***
  UART/USART node address (7-bit long max).

69.2 UARTEx Firmware driver API description

69.2.1 UART peripheral extended features

69.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_RS485Ex_Init() API follows the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_RS485Ex_Init()***

69.2.3 IO operation functions

This section contains the following APIs:

- ***HAL_UARTEx_WakeupCallback()***
- ***HAL_UARTEx_RxFifoFullCallback()***
- ***HAL_UARTEx_TxFifoEmptyCallback()***

69.2.4 Peripheral Control functions

This section provides the following functions:

- HAL_UARTEx_EnableClockStopMode() API enables the UART clock (HSI or LSE only) during stop mode
- HAL_UARTEx_DisableClockStopMode() API disables the above functionality
- HAL_MultiProcessorEx_AddressLength_Set() API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- HAL_UARTEx_StopModeWakeUpSourceConfig() API defines the wake-up from stop mode trigger: address match, Start Bit detection or RXNE bit status.
- HAL_UARTEx_EnableStopMode() API enables the UART to wake up the MCU from stop mode
- HAL_UARTEx_DisableStopMode() API disables the above functionality
- HAL_UARTEx_WakeupCallback() called upon UART wakeup interrupt
- HAL_UARTEx_EnableSPISlaveMode() API enables the SPI slave mode
- HAL_UARTEx_DisableSPISlaveMode() API disables the SPI slave mode
- HAL_UARTEx_ConfigNSS API configures the Slave Select input pin (NSS)
- HAL_UARTEx_EnableFifoMode() API enables the FIFO mode
- HAL_UARTEx_DisableFifoMode() API disables the FIFO mode
- HAL_UARTEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_UARTEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- ***HAL_MultiProcessorEx_AddressLength_Set()***
- ***HAL_UARTEx_StopModeWakeUpSourceConfig()***
- ***HAL_UARTEx_EnableStopMode()***
- ***HAL_UARTEx_DisableStopMode()***
- ***HAL_UARTEx_EnableSlaveMode()***
- ***HAL_UARTEx_DisableSlaveMode()***
- ***HAL_UARTEx_ConfigNSS()***
- ***HAL_UARTEx_EnableFifoMode()***
- ***HAL_UARTEx_DisableFifoMode()***
- ***HAL_UARTEx_SetTxFifoThreshold()***
- ***HAL_UARTEx_SetRxFifoThreshold()***

69.2.5 Detailed description of functions

HAL_RS485Ex_Init

Function name	**HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)**
Function description	Initialize the RS485 Driver enable feature according to the specified parameters in the UART_InitTypeDef and creates the associated handle.
Parameters	• **huart:** UART handle. • **Polarity:** Select the driver enable polarity. This parameter can be one of the following values: – UART_DE_POLARITY_HIGH DE signal is active high – UART_DE_POLARITY_LOW DE signal is active low • **AssertionTime:** Driver Enable assertion time: 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate) • **DeassertionTime:** Driver Enable deassertion time: 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).
Return values	• **HAL:** status

HAL_UARTEx_WakeupCallback

Function name	**void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)**
Function description	UART wakeup from Stop mode callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UARTEx_RxFifoFullCallback

Function name	**void HAL_UARTEx_RxFifoFullCallback (UART_HandleTypeDef * huart)**
Function description	UART RX Fifo full callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UARTEx_TxFifoEmptyCallback

Function name	**void HAL_UARTEx_TxFifoEmptyCallback (UART_HandleTypeDef * huart)**
Function description	UART TX Fifo empty callback.

Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UARTEx_StopModeWakeUpSourceConfig

Function name	**HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)**
Function description	Set Wakeup from Stop mode interrupt flag selection.
Parameters	• **huart:** UART handle. • **WakeUpSelection:** Address match, Start Bit detection or RXNE/RXFNE bit status. This parameter can be one of the following values: – UART_WAKEUP_ON_ADDRESS – UART_WAKEUP_ON_STARTBIT – UART_WAKEUP_ON_READDATA_NONEMPTY
Return values	• **HAL:** status
Notes	• It is the application responsibility to enable the interrupt used as usart_wkup interrupt source before entering low-power mode.

HAL_UARTEx_EnableStopMode

Function name	**HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (UART_HandleTypeDef * huart)**
Function description	Enable UART Stop Mode.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• The UART is able to wake up the MCU from Stop 1 mode as long as UART clock is HSI or LSE.

HAL_UARTEx_DisableStopMode

Function name	**HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (UART_HandleTypeDef * huart)**
Function description	Disable UART Stop Mode.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_MultiProcessorEx_AddressLength_Set

Function name	**HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)**
Function description	By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses

	detection; this API allows to enable longer addresses detection (6-, 7- or 8-bit long).
Parameters	• **huart:** UART handle. • **AddressLength:** This parameter can be one of the following values: – UART_ADDRESS_DETECT_4B 4-bit long address – UART_ADDRESS_DETECT_7B 6-, 7- or 8-bit long address
Return values	• **HAL:** status
Notes	• Addresses detection lengths are: 6-bit address detection in 7-bit data mode, 7-bit address detection in 8-bit data mode, 8-bit address detection in 9-bit data mode.

HAL_UARTEx_EnableSlaveMode

Function name	**HAL_StatusTypeDef HAL_UARTEx_EnableSlaveMode (UART_HandleTypeDef * huart)**
Function description	Enable the SPI slave mode.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• When the UART operates in SPI slave mode, it handles data flow using the serial interface clock derived from the external SCLK signal provided by the external master SPI device. • In SPI slave mode, the UART must be enabled before starting the master communications (or between frames while the clock is stable). Otherwise, if the UART slave is enabled while the master is in the middle of a frame, it will become desynchronized with the master. • The data register of the slave needs to be ready before the first edge of the communication clock or before the end of the ongoing communication, otherwise the SPI slave will transmit zeros.

HAL_UARTEx_DisableSlaveMode

Function name	**HAL_StatusTypeDef HAL_UARTEx_DisableSlaveMode (UART_HandleTypeDef * huart)**
Function description	Disable the SPI slave mode.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_UARTEx_ConfigNSS

Function name	**HAL_StatusTypeDef HAL_UARTEx_ConfigNSS (UART_HandleTypeDef * huart, uint32_t NSSConfig)**
Function description	Configure the Slave Select input pin (NSS).
Parameters	• **huart:** UART handle. • **NSSConfig:** NSS configuration. This parameter can be one

	of the following values: – UART_NSS_HARD – UART_NSS_SOFT
Return values	• **HAL:** status
Notes	• Software NSS management: SPI slave will always be selected and NSS input pin will be ignored. • Hardware NSS management: the SPI slave selection depends on NSS input pin. The slave is selected when NSS is low and deselected when NSS is high.

HAL_UARTEx_EnableFifoMode

Function name	**HAL_StatusTypeDef HAL_UARTEx_EnableFifoMode (UART_HandleTypeDef * huart)**
Function description	Enable the FIFO mode.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_UARTEx_DisableFifoMode

Function name	**HAL_StatusTypeDef HAL_UARTEx_DisableFifoMode (UART_HandleTypeDef * huart)**
Function description	Disable the FIFO mode.
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status

HAL_UARTEx_SetTxFifoThreshold

Function name	**HAL_StatusTypeDef HAL_UARTEx_SetTxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)**
Function description	Set the TXFIFO threshold.
Parameters	• **huart:** UART handle. • **Threshold:** TX FIFO threshold value This parameter can be one of the following values: – UART_TXFIFO_THRESHOLD_1_8 – UART_TXFIFO_THRESHOLD_1_4 – UART_TXFIFO_THRESHOLD_1_2 – UART_TXFIFO_THRESHOLD_3_4 – UART_TXFIFO_THRESHOLD_7_8 – UART_TXFIFO_THRESHOLD_8_8
Return values	• **HAL:** status

HAL_UARTEx_SetRxFifoThreshold

Function name	**HAL_StatusTypeDef HAL_UARTEx_SetRxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)**
Function description	Set the RXFIFO threshold.

Parameters	• **huart:** UART handle. • **Threshold:** RX FIFO threshold value This parameter can be one of the following values: – UART_RXFIFO_THRESHOLD_1_8 – UART_RXFIFO_THRESHOLD_1_4 – UART_RXFIFO_THRESHOLD_1_2 – UART_RXFIFO_THRESHOLD_3_4 – UART_RXFIFO_THRESHOLD_7_8 – UART_RXFIFO_THRESHOLD_8_8
Return values	• **HAL:** status

69.3 UARTEx Firmware driver defines

69.3.1 UARTEx

UARTEx FIFO mode

UART_FIFOMODE_DISABLE	FIFO mode disable
UART_FIFOMODE_ENABLE	FIFO mode enable

UARTEx RXFIFO threshold level

UART_RXFIFO_THRESHOLD_1_8	RXFIFO FIFO reaches 1/8 of its depth
UART_RXFIFO_THRESHOLD_1_4	RXFIFO FIFO reaches 1/4 of its depth
UART_RXFIFO_THRESHOLD_1_2	RXFIFO FIFO reaches 1/2 of its depth
UART_RXFIFO_THRESHOLD_3_4	RXFIFO FIFO reaches 3/4 of its depth
UART_RXFIFO_THRESHOLD_7_8	RXFIFO FIFO reaches 7/8 of its depth
UART_RXFIFO_THRESHOLD_8_8	RXFIFO FIFO becomes full

UARTEx Synchronous Slave mode

UART_SLAVEMODE_DISABLE	USART SPI Slave Mode Enable
UART_SLAVEMODE_ENABLE	USART SPI Slave Mode Disable

UARTEx Slave Select Management

UART_NSS_HARD	SPI slave selection depends on NSS input pin
UART_NSS_SOFT	SPI slave is always selected and NSS input pin is ignored

UARTEx TXFIFO threshold level

UART_TXFIFO_THRESHOLD_1_8	TXFIFO reaches 1/8 of its depth
UART_TXFIFO_THRESHOLD_1_4	TXFIFO reaches 1/4 of its depth
UART_TXFIFO_THRESHOLD_1_2	TXFIFO reaches 1/2 of its depth
UART_TXFIFO_THRESHOLD_3_4	TXFIFO reaches 3/4 of its depth
UART_TXFIFO_THRESHOLD_7_8	TXFIFO reaches 7/8 of its depth
UART_TXFIFO_THRESHOLD_8_8	TXFIFO becomes empty

UARTEx WakeUp Address Length

UART_ADDRESS_DETECT_4B	4-bit long wake-up address

UART_ADDRESS_DETECT_7B 7-bit long wake-up address

UARTEx Word Length

UART_WORDLENGTH_7B 7-bit long UART frame

UART_WORDLENGTH_8B 8-bit long UART frame

UART_WORDLENGTH_9B 9-bit long UART frame

70 HAL USART Generic Driver

70.1 USART Firmware driver registers structures

70.1.1 USART_InitTypeDef

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t CLKLastBit***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t USART_InitTypeDef::BaudRate***
  This member configures the Usart communication baud rate. The baud rate is computed using the following formula: Baud Rate Register[15:4] = ((2 * fclk_pres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * fclk_pres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where fclk_pres is the USART input clock frequency (fclk) divided by a prescaler.
  Note: Oversampling by 8 is systematically applied to achieve high baud rates.
- ***uint32_t USART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USARTEx_Word_Length***.
- ***uint32_t USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_Stop_Bits***.
- ***uint32_t USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_Parity***
  Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t USART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_Mode***.
- ***uint32_t USART_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_Clock_Polarity***.
- ***uint32_t USART_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_Clock_Phase***.
- ***uint32_t USART_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_Last_Bit***.

- ***uint32_t USART_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***USART_ClockPrescaler***.

70.1.2 __USART_HandleTypeDef

Data Fields

- ***USART_TypeDef * Instance***
- ***USART_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t FifoMode***
- ***uint32_t SlaveMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_USART_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* __USART_HandleTypeDef::Instance***
  USART registers base address
- ***USART_InitTypeDef __USART_HandleTypeDef::Init***
  USART communication parameters
- ***uint8_t* __USART_HandleTypeDef::pTxBuffPtr***
  Pointer to USART Tx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::TxXferSize***
  USART Tx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::TxXferCount***
  USART Tx Transfer Counter
- ***uint8_t* __USART_HandleTypeDef::pRxBuffPtr***
  Pointer to USART Rx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::RxXferSize***
  USART Rx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::RxXferCount***
  USART Rx Transfer Counter
- ***uint16_t __USART_HandleTypeDef::Mask***
  USART Rx RDR register mask
- ***uint16_t __USART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __USART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution

- ***uint32_t __USART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode is being used. This parameter can be a value of ***USARTEx_FIFO_mode***.
- ***uint32_t __USART_HandleTypeDef::SlaveMode***
  Specifies if the UART SPI Slave mode is being used. This parameter can be a value of ***USARTEx_Slave_Mode***.
- ***void(* __USART_HandleTypeDef::RxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Rx IRQ handler
- ***void(* __USART_HandleTypeDef::TxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmatx***
  USART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmarx***
  USART Rx DMA Handle parameters
- ***HAL_LockTypeDef __USART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_USART_StateTypeDef __USART_HandleTypeDef::State***
  USART communication state
- ***__IO uint32_t __USART_HandleTypeDef::ErrorCode***
  USART Error code

70.2 USART Firmware driver API description

70.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure (eg. USART_HandleTypeDef husart).
2. Initialize the USART low level resources by implementing the HAL_USART_MspInit() API:
   - Enable the USARTx interface clock.
   - USART pins configuration:
     - Enable the clock for the USART GPIOs.
     - Configure these USART pins as alternate function pull-up.
   - NVIC configuration if you need to use interrupt process (HAL_USART_Transmit_IT(), HAL_USART_Receive_IT() and HAL_USART_TransmitReceive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - USART interrupts handling: The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_USART_ENABLE_IT() and __HAL_USART_DISABLE_IT() inside the transmit and receive process.
   - DMA Configuration if you need to use DMA process (HAL_USART_Transmit_DMA() HAL_USART_Receive_DMA() and HAL_USART_TransmitReceive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the USART DMA Tx/Rx handle.

– Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.

3. Program the Baud Rate, Word Length, Stop Bit, Parity, and Mode (Receiver/Transmitter) in the husart handle Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_USART_MspInit(&husart) API.

To configure and enable/disable the USART to wake up the MCU from stop mode, resort to UART API's HAL_UARTEx_StopModeWakeUpSourceConfig(), HAL_UARTEx_EnableStopMode() and HAL_UARTEx_DisableStopMode() in casting the USART handle to UART type UART_HandleTypeDef.

70.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
  – Baud Rate
  – Word Length
  – Stop Bit
  – Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  – USART polarity
  – USART phase
  – USART LastBit
  – Receiver/transmitter modes

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

This section contains the following APIs:

- ***HAL_USART_Init()***
- ***HAL_USART_DeInit()***
- ***HAL_USART_MspInit()***
- ***HAL_USART_MspDeInit()***

70.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the USART synchronous data transfers.

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
   – Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   – No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive

process The HAL_USART_ErrorCallback()user callback will be executed when a communication error is detected

2. Blocking mode APIs are:
   - HAL_USART_Transmit()in simplex mode
   - HAL_USART_Receive() in full duplex receive only
   - HAL_USART_TransmitReceive() in full duplex mode
3. Non-Blocking mode APIs with Interrupt are:
   - HAL_USART_Transmit_IT()in simplex mode
   - HAL_USART_Receive_IT() in full duplex receive only
   - HAL_USART_TransmitReceive_IT()in full duplex mode
   - HAL_USART_IRQHandler()
4. No-Blocking mode APIs with DMA are:
   - HAL_USART_Transmit_DMA()in simplex mode
   - HAL_USART_Receive_DMA() in full duplex receive only
   - HAL_USART_TransmitReceive_DMA() in full duplex mode
   - HAL_USART_DMAPause()
   - HAL_USART_DMAResume()
   - HAL_USART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non_Blocking mode:
   - HAL_USART_TxCpltCallback()
   - HAL_USART_RxCpltCallback()
   - HAL_USART_TxHalfCpltCallback()
   - HAL_USART_RxHalfCpltCallback()
   - HAL_USART_ErrorCallback()
   - HAL_USART_TxRxCpltCallback()
6. Non-Blocking mode transfers could be aborted using Abort APIs:
   - HAL_USART_Abort()
   - HAL_USART_Abort_IT()
7. For Abort services based on interrupts (HAL_USART_Abort_IT), a Abort Complete Callbacks is provided: HAL_USART_AbortCpltCallback()
8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows:
   - Error is considered as Recoverable and non blocking: Transfer could go till end, but error severity is to be evaluated by user: this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed. Transfer is kept ongoing on USART side. If user wants to abort it, Abort services should be called by user.
   - Error is considered as Blocking: Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_USART_Transmit()***
- ***HAL_USART_Receive()***
- ***HAL_USART_TransmitReceive()***
- ***HAL_USART_Transmit_IT()***
- ***HAL_USART_Receive_IT()***
- ***HAL_USART_TransmitReceive_IT()***
- ***HAL_USART_Transmit_DMA()***
- ***HAL_USART_Receive_DMA()***
- ***HAL_USART_TransmitReceive_DMA()***

- *HAL_USART_DMAPause()*
- *HAL_USART_DMAResume()*
- *HAL_USART_DMAStop()*
- *HAL_USART_Abort()*
- *HAL_USART_Abort_IT()*
- *HAL_USART_IRQHandler()*
- *HAL_USART_TxCpltCallback()*
- *HAL_USART_TxHalfCpltCallback()*
- *HAL_USART_RxCpltCallback()*
- *HAL_USART_RxHalfCpltCallback()*
- *HAL_USART_TxRxCpltCallback()*
- *HAL_USART_ErrorCallback()*
- *HAL_USART_AbortCpltCallback()*

70.2.4 Peripheral State and Error functions

This subsection provides functions allowing to:

- Return the USART handle state
- Return the USART handle error code

This section contains the following APIs:

- *HAL_USART_GetState()*
- *HAL_USART_GetError()*

70.2.5 Detailed description of functions

HAL_USART_Init

Function name	**HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)**
Function description	Initialize the USART mode according to the specified parameters in the USART_InitTypeDef and initialize the associated handle.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USART_DeInit

Function name	**HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)**
Function description	DeInitialize the USART peripheral.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USART_MspInit

Function name	**void HAL_USART_MspInit (USART_HandleTypeDef * husart)**
Function description	Initialize the USART MSP.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_MspDeInit

Function name	**void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)**
Function description	DeInitialize the USART MSP.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_Transmit

Function name	**HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)**
Function description	Simplex send an amount of data in blocking mode.
Parameters	• **husart:** USART handle. • **pTxData:** Pointer to data buffer. • **Size:** Amount of data to be sent. • **Timeout:** Timeout duration.
Return values	• **HAL:** status

HAL_USART_Receive

Function name	**HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **husart:** USART handle. • **pRxData:** Pointer to data buffer. • **Size:** Amount of data to be received. • **Timeout:** Timeout duration.
Return values	• **HAL:** status
Notes	• To receive synchronous data, dummy data are simultaneously transmitted.

HAL_USART_TransmitReceive

Function name	**HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)**
Function description	Full-Duplex Send and Receive an amount of data in blocking mode.
Parameters	• **husart:** USART handle. • **pTxData:** pointer to TX data buffer. • **pRxData:** pointer to RX data buffer. • **Size:** amount of data to be sent (same amount to be received). • **Timeout:** Timeout duration.

Return values	• **HAL:** status

HAL_USART_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)**
Function description	Send an amount of data in interrupt mode.
Parameters	• **husart:** USART handle. • **pTxData:** pointer to data buffer. • **Size:** amount of data to be sent.
Return values	• **HAL:** status

HAL_USART_Receive_IT

Function name	**HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **husart:** USART handle. • **pRxData:** pointer to data buffer. • **Size:** amount of data to be received.
Return values	• **HAL:** status
Notes	• To receive synchronous data, dummy data are simultaneously transmitted.

HAL_USART_TransmitReceive_IT

Function name	**HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**
Function description	Full-Duplex Send and Receive an amount of data in interrupt mode.
Parameters	• **husart:** USART handle. • **pTxData:** pointer to TX data buffer. • **pRxData:** pointer to RX data buffer. • **Size:** amount of data to be sent (same amount to be received).
Return values	• **HAL:** status

HAL_USART_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)**
Function description	Send an amount of data in DMA mode.
Parameters	• **husart:** USART handle.

	• **pTxData:** pointer to data buffer. • **Size:** amount of data to be sent.
Return values	• **HAL:** status

HAL_USART_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)**
Function description	Receive an amount of data in DMA mode.
Parameters	• **husart:** USART handle. • **pRxData:** pointer to data buffer. • **Size:** amount of data to be received.
Return values	• **HAL:** status
Notes	• When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position). • The USART DMA transmit channel must be configured in order to generate the clock for the slave.

HAL_USART_TransmitReceive_DMA

Function name	**HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**
Function description	Full-Duplex Transmit Receive an amount of data in non-blocking mode.
Parameters	• **husart:** USART handle. • **pTxData:** pointer to TX data buffer. • **pRxData:** pointer to RX data buffer. • **Size:** amount of data to be received/sent.
Return values	• **HAL:** status
Notes	• When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

HAL_USART_DMAPause

Function name	**HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)**
Function description	Pause the DMA Transfer.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USART_DMAResume

Function name	**HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)**

Function description	Resume the DMA Transfer.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USART_DMAStop

Function name	**HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)**
Function description	Stop the DMA Transfer.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USART_Abort

Function name	**HAL_StatusTypeDef HAL_USART_Abort (USART_HandleTypeDef * husart)**
Function description	Abort ongoing transfers (blocking mode).
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode: when exiting function, Abort is considered as completed.

HAL_USART_Abort_IT

Function name	**HAL_StatusTypeDef HAL_USART_Abort_IT (USART_HandleTypeDef * husart)**
Function description	Abort ongoing transfers (Interrupt mode).
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations: Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_USART_IRQHandler

Function name	**void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)**
Function description	Handle USART interrupt request.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_TxHalfCpltCallback

Function name	**void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)**
Function description	Tx Half Transfer completed callback.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_TxCpltCallback

Function name	**void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)**
Function description	Tx Transfer completed callback.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_RxCpltCallback

Function name	**void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)**
Function description	Rx Transfer completed callback.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_RxHalfCpltCallback

Function name	**void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)**
Function description	Rx Half Transfer completed callback.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_TxRxCpltCallback

Function name	**void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)**
Function description	Tx/Rx Transfers completed callback for the non-blocking process.

Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_ErrorCallback

Function name	**void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)**
Function description	USART error callback.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_AbortCpltCallback

Function name	**void HAL_USART_AbortCpltCallback (USART_HandleTypeDef * husart)**
Function description	USART Abort Complete callback.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_GetState

Function name	**HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)**
Function description	Return the USART handle state.
Parameters	• **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.
Return values	• **USART:** handle state

HAL_USART_GetError

Function name	**uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)**
Function description	Return the USART error code.
Parameters	• **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.
Return values	• **USART:** handle Error Code

70.3 USART Firmware driver defines

70.3.1 USART

USART Clock

USART_CLOCK_DISABLE	USART clock disable

USART_CLOCK_ENABLE	USART clock enable

USART Clock Prescaler

USART_PRESCALER_DIV1	fclk_pres = fclk
USART_PRESCALER_DIV2	fclk_pres = fclk/2
USART_PRESCALER_DIV4	fclk_pres = fclk/4
USART_PRESCALER_DIV6	fclk_pres = fclk/6
USART_PRESCALER_DIV8	fclk_pres = fclk/8
USART_PRESCALER_DIV10	fclk_pres = fclk/10
USART_PRESCALER_DIV12	fclk_pres = fclk/12
USART_PRESCALER_DIV16	fclk_pres = fclk/16
USART_PRESCALER_DIV32	fclk_pres = fclk/32
USART_PRESCALER_DIV64	fclk_pres = fclk/64
USART_PRESCALER_DIV128	fclk_pres = fclk/128
USART_PRESCALER_DIV256	fclk_pres = fclk/256

USART Clock Phase

USART_PHASE_1EDGE	USART frame phase on first clock transition
USART_PHASE_2EDGE	USART frame phase on second clock transition

USART Clock Polarity

USART_POLARITY_LOW	Driver enable signal is active high
USART_POLARITY_HIGH	Driver enable signal is active low

USART Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__: USART handle.

Return value:

- None

__HAL_USART_GET_FLAG

Description:

- Check whether the specified USART flag is set or not.

Parameters:

- __HANDLE__: specifies the USART Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - USART_FLAG_TXFT TXFIFO threshold flag
  - USART_FLAG_RXFT RXFIFO threshold flag
  - USART_FLAG_RXFF RXFIFO Full flag

- – USART_FLAG_TXFE TXFIFO Empty flag
- – USART_FLAG_REACK Receive enable acknowledge flag
- – USART_FLAG_TEACK Transmit enable acknowledge flag
- – USART_FLAG_BUSY Busy flag
- – USART_FLAG_UDR SPI slave underrun error flag
- – USART_FLAG_TXE Transmit data register empty flag
- – USART_FLAG_TXFNF TXFIFO not full flag
- – USART_FLAG_TC Transmission Complete flag
- – USART_FLAG_RXNE Receive data register not empty flag
- – USART_FLAG_RXFNE RXFIFO not empty flag
- – USART_FLAG_IDLE Idle Line detection flag
- – USART_FLAG_ORE OverRun Error flag
- – USART_FLAG_NE Noise Error flag
- – USART_FLAG_FE Framing Error flag
- – USART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_USART_CLEAR_FLAG

Description:

- Clear the specified USART pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - – USART_CLEAR_PEF Parity Error Clear Flag
  - – USART_CLEAR_FEF Framing Error Clear Flag
  - – USART_CLEAR_NEF Noise detected Clear Flag
  - – USART_CLEAR_OREF Overrun Error Clear Flag
  - – USART_CLEAR_IDLEF IDLE line detected Clear Flag
  - – USART_CLEAR_TXFECF TXFIFO empty clear Flag
  - – USART_CLEAR_TCF Transmission Complete Clear Flag
  - – USART_CLEAR_UDRF SPI slave underrun error Clear Flag

Return value:

- None

__HAL_USART_CLEAR_PEFLAG

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_FEFLAG

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_NEFLAG

Description:

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_OREFLAG

Description:

- Clear the USART ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_IDLEFLAG

Description:

- Clear the USART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_TXFECF

Description:

- Clear the USART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_UDRFLAG

Description:

- Clear SPI slave underrun error flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE_IT

Description:

- Enable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_PE Parity Error interrupt
  - USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_DISABLE_IT

Description:

- Disable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to disable. This parameter can be one of the following values:

- USART_IT_RXFF RXFIFO Full interrupt
- USART_IT_TXFE TXFIFO Empty interrupt
- USART_IT_RXFT RXFIFO threshold interrupt
- USART_IT_TXFT TXFIFO threshold interrupt
- USART_IT_TXE Transmit Data Register empty interrupt
- USART_IT_TXFNF TX FIFO not full interrupt
- USART_IT_TC Transmission complete interrupt
- USART_IT_RXNE Receive Data register not empty interrupt
- USART_IT_RXFNE RXFIFO not empty interrupt
- USART_IT_IDLE Idle line detection interrupt
- USART_IT_PE Parity Error interrupt
- USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_GET_IT

Description:

- Check whether the specified USART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt

- USART_IT_FE Framing Error interrupt
- USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_GET_IT_SOURCE

Description:

- Check whether the specified USART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt
  - USART_IT_FE Framing Error interrupt
  - USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_CLEAR_IT

Description:

- Clear the specified USART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - USART_CLEAR_PEF Parity Error Clear

Flag

– USART_CLEAR_FEF Framing Error Clear Flag
– USART_CLEAR_NEF Noise detected Clear Flag
– USART_CLEAR_OREF Overrun Error Clear Flag
– USART_CLEAR_IDLEF IDLE line detected Clear Flag
– USART_CLEAR_TXFECF TXFIFO empty clear Flag
– USART_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_USART_SEND_REQ

Description:

- Set a specific USART request flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __REQ__: specifies the request flag to set. This parameter can be one of the following values:
  – USART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  – USART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_ ENABLE

Description:

- Enable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_ DISABLE

Description:

- Disable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE

Description:

- Enable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_DISABLE

Description:

- Disable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

USART Flags

USART_FLAG_TXFT	USART TXFIFO threshold flag
USART_FLAG_RXFT	USART RXFIFO threshold flag
USART_FLAG_RXFF	USART RXFIFO Full flag
USART_FLAG_TXFE	USART TXFIFO Empty flag
USART_FLAG_TXE	USART transmit data register empty
USART_FLAG_TXFNF	USART TXFIFO not full
USART_FLAG_RXNE	USART read data register not empty
USART_FLAG_RXFNE	USART RXFIFO not empty
USART_FLAG_REACK	USART receive enable acknowledge flag
USART_FLAG_TEACK	USART transmit enable acknowledge flag
USART_FLAG_BUSY	USART busy flag
USART_FLAG_TC	USART transmission complete
USART_FLAG_IDLE	USART idle flag
USART_FLAG_ORE	USART overrun error
USART_FLAG_NE	USART noise error
USART_FLAG_FE	USART frame error
USART_FLAG_PE	USART parity error
USART_FLAG_UDR	SPI slave underrun error flag

USART Interruption Flags Mask

USART_IT_MASK	USART interruptions flags mask

USART Interrupts Definition

USART_IT_PE	USART parity error interruption
USART_IT_TXFNF	USART TX FIFO not full interruption
USART_IT_RXFNE	USART RXFIFO not empty interruption
USART_IT_RXFF	USART RXFIFO full interruption
USART_IT_TXFE	USART TXFIFO empty interruption

USART_IT_RXFT	USART RXFIFO threshold reached interruption
USART_IT_TXFT	USART TXFIFO threshold reached interruption
USART_IT_TXE	USART transmit data register empty interruption
USART_IT_TC	USART transmission complete interruption
USART_IT_RXNE	USART read data register not empty interruption
USART_IT_IDLE	USART idle interruption
USART_IT_ERR	USART error interruption
USART_IT_ORE	USART overrun error interruption
USART_IT_NE	USART noise error interruption
USART_IT_FE	USART frame error interruption

USART Interruption Clear Flags

USART_CLEAR_PEF	Parity Error Clear Flag
USART_CLEAR_FEF	Framing Error Clear Flag
USART_CLEAR_NEF	Noise detected Clear Flag
USART_CLEAR_OREF	OverRun Error Clear Flag
USART_CLEAR_IDLEF	IDLE line detected Clear Flag
USART_CLEAR_TCF	Transmission Complete Clear Flag
USART_CLEAR_TXFECF	TXFIFO Empty Clear Flag
USART_CLEAR_UDRF	SPI slave underrun error Clear Flag

USART Last Bit

USART_LASTBIT_DISABLE	USART frame last data bit clock pulse not output to SCLK pin
USART_LASTBIT_ENABLE	USART frame last data bit clock pulse output to SCLK pin

USART Mode

USART_MODE_RX	RX mode
USART_MODE_TX	TX mode
USART_MODE_TX_RX	RX and TX mode

USART Over Sampling

USART_OVERSAMPLING_16	Oversampling by 16
USART_OVERSAMPLING_8	Oversampling by 8

USART Parity

USART_PARITY_NONE	No parity
USART_PARITY_EVEN	Even parity
USART_PARITY_ODD	Odd parity

USART Request Parameters

USART_RXDATA_FLUSH_REQUEST	Receive Data flush Request
USART_TXDATA_FLUSH_REQUEST	Transmit data flush Request

USART Number of Stop Bits

USART_STOPBITS_0_5	USART frame with 0.5 stop bit
USART_STOPBITS_1	USART frame with 1 stop bit
USART_STOPBITS_1_5	USART frame with 1.5 stop bits
USART_STOPBITS_2	USART frame with 2 stop bits

71 HAL USART Extension Driver

71.1 USARTEx Firmware driver API description

71.1.1 USART peripheral extended features

71.1.2 IO operation functions

This section contains the following APIs:

- *HAL_USARTEx_RxFifoFullCallback()*
- *HAL_USARTEx_TxFifoEmptyCallback()*

71.1.3 Peripheral Control functions

This section provides the following functions:

- HAL_USARTEx_EnableSPISlaveMode() API enables the SPI slave mode
- HAL_USARTEx_DisableSPISlaveMode() API disables the SPI slave mode
- HAL_USARTEx_ConfigNSS API configures the Slave Select input pin (NSS)
- HAL_USARTEx_EnableFifoMode() API enables the FIFO mode
- HAL_USARTEx_DisableFifoMode() API disables the FIFO mode
- HAL_USARTEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_USARTEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- *HAL_USARTEx_EnableSlaveMode()*
- *HAL_USARTEx_DisableSlaveMode()*
- *HAL_USARTEx_ConfigNSS()*
- *HAL_USARTEx_EnableFifoMode()*
- *HAL_USARTEx_DisableFifoMode()*
- *HAL_USARTEx_SetTxFifoThreshold()*
- *HAL_USARTEx_SetRxFifoThreshold()*

71.1.4 Detailed description of functions

HAL_USARTEx_RxFifoFullCallback

Function name	**void HAL_USARTEx_RxFifoFullCallback (USART_HandleTypeDef * husart)**
Function description	USART RX Fifo full callback.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USARTEx_TxFifoEmptyCallback

Function name	**void HAL_USARTEx_TxFifoEmptyCallback (USART_HandleTypeDef * husart)**
Function description	USART TX Fifo empty callback.
Parameters	• **husart:** USART handle.

Return values	• **None:**

HAL_USARTEx_EnableSlaveMode

Function name	**HAL_StatusTypeDef HAL_USARTEx_EnableSlaveMode (USART_HandleTypeDef * husart)**
Function description	Enable the SPI slave mode.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status
Notes	• When the USART operates in SPI slave mode, it handles data flow using the serial interface clock derived from the external SCLK signal provided by the external master SPI device. • In SPI slave mode, the USART must be enabled before starting the master communications (or between frames while the clock is stable). Otherwise, if the USART slave is enabled while the master is in the middle of a frame, it will become desynchronized with the master. • The data register of the slave needs to be ready before the first edge of the communication clock or before the end of the ongoing communication, otherwise the SPI slave will transmit zeros.

HAL_USARTEx_DisableSlaveMode

Function name	**HAL_StatusTypeDef HAL_USARTEx_DisableSlaveMode (USART_HandleTypeDef * husart)**
Function description	Disable the SPI slave mode.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USARTEx_ConfigNSS

Function name	**HAL_StatusTypeDef HAL_USARTEx_ConfigNSS (USART_HandleTypeDef * husart, uint32_t NSSConfig)**
Function description	Configure the Slave Select input pin (NSS).
Parameters	• **husart:** USART handle. • **NSSConfig:** NSS configuration. This parameter can be one of the following values: – USART_NSS_HARD – USART_NSS_SOFT
Return values	• **HAL:** status
Notes	• Software NSS management: SPI slave will always be selected and NSS input pin will be ignored. • Hardware NSS management: the SPI slave selection depends on NSS input pin. The slave is selected when NSS is low and deselected when NSS is high.

HAL_USARTEx_EnableFifoMode

Function name	**HAL_StatusTypeDef HAL_USARTEx_EnableFifoMode (USART_HandleTypeDef * husart)**
Function description	Enable the FIFO mode.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USARTEx_DisableFifoMode

Function name	**HAL_StatusTypeDef HAL_USARTEx_DisableFifoMode (USART_HandleTypeDef * husart)**
Function description	Disable the FIFO mode.
Parameters	• **husart:** USART handle.
Return values	• **HAL:** status

HAL_USARTEx_SetTxFifoThreshold

Function name	**HAL_StatusTypeDef HAL_USARTEx_SetTxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)**
Function description	Set the TXFIFO threshold.
Parameters	• **husart:** USART handle. • **Threshold:** TX FIFO threshold value This parameter can be one of the following values: – USART_TXFIFO_THRESHOLD_1_8 – USART_TXFIFO_THRESHOLD_1_4 – USART_TXFIFO_THRESHOLD_1_2 – USART_TXFIFO_THRESHOLD_3_4 – USART_TXFIFO_THRESHOLD_7_8 – USART_TXFIFO_THRESHOLD_8_8
Return values	• **HAL:** status

HAL_USARTEx_SetRxFifoThreshold

Function name	**HAL_StatusTypeDef HAL_USARTEx_SetRxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)**
Function description	Set the RXFIFO threshold.
Parameters	• **husart:** USART handle. • **Threshold:** RX FIFO threshold value This parameter can be one of the following values: – USART_RXFIFO_THRESHOLD_1_8 – USART_RXFIFO_THRESHOLD_1_4 – USART_RXFIFO_THRESHOLD_1_2 – USART_RXFIFO_THRESHOLD_3_4 – USART_RXFIFO_THRESHOLD_7_8 – USART_RXFIFO_THRESHOLD_8_8
Return values	• **HAL:** status

71.2 USARTEx Firmware driver defines

71.2.1 USARTEx

USARTEx FIFO mode

USART_FIFOMODE_DISABLE FIFO mode disable

USART_FIFOMODE_ENABLE FIFO mode enable

USARTEx RXFIFO threshold level

USART_RXFIFO_THRESHOLD_1_8 RXFIFO FIFO reaches 1/8 of its depth

USART_RXFIFO_THRESHOLD_1_4 RXFIFO FIFO reaches 1/4 of its depth

USART_RXFIFO_THRESHOLD_1_2 RXFIFO FIFO reaches 1/2 of its depth

USART_RXFIFO_THRESHOLD_3_4 RXFIFO FIFO reaches 3/4 of its depth

USART_RXFIFO_THRESHOLD_7_8 RXFIFO FIFO reaches 7/8 of its depth

USART_RXFIFO_THRESHOLD_8_8 RXFIFO FIFO becomes full

USARTEx Synchronous Slave mode

USART_SLAVEMODE_DISABLE USART SPI Slave Mode Enable

USART_SLAVEMODE_ENABLE USART SPI Slave Mode Disable

USARTEx Slave Select Management

USART_NSS_HARD SPI slave selection depends on NSS input pin

USART_NSS_SOFT SPI slave is always selected and NSS input pin is ignored

USARTEx TXFIFO threshold level

USART_TXFIFO_THRESHOLD_1_8 TXFIFO reaches 1/8 of its depth

USART_TXFIFO_THRESHOLD_1_4 TXFIFO reaches 1/4 of its depth

USART_TXFIFO_THRESHOLD_1_2 TXFIFO reaches 1/2 of its depth

USART_TXFIFO_THRESHOLD_3_4 TXFIFO reaches 3/4 of its depth

USART_TXFIFO_THRESHOLD_7_8 TXFIFO reaches 7/8 of its depth

USART_TXFIFO_THRESHOLD_8_8 TXFIFO becomes empty

USARTEx Word Length

USART_WORDLENGTH_7B 7-bit long USART frame

USART_WORDLENGTH_8B 8-bit long USART frame

USART_WORDLENGTH_9B 9-bit long USART frame

72 HAL WWDG Generic Driver

72.1 WWDG Firmware driver registers structures

72.1.1 WWDG_InitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Window***
- ***uint32_t Counter***
- ***uint32_t EWIMode***

Field Documentation

- ***uint32_t WWDG_InitTypeDef::Prescaler***
  Specifies the prescaler value of the WWDG. This parameter can be a value of ***WWDG_Prescaler***
- ***uint32_t WWDG_InitTypeDef::Window***
  Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::Counter***
  Specifies the WWDG free-running downcounter value. This parameter must be a number between Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::EWIMode***
  Specifies if WWDG Early Wakeup Interupt is enable or not. This parameter can be a value of ***WWDG_EWI_Mode***

72.1.2 WWDG_HandleTypeDef

Data Fields

- ***WWDG_TypeDef * Instance***
- ***WWDG_InitTypeDef Init***

Field Documentation

- ***WWDG_TypeDef* WWDG_HandleTypeDef::Instance***
  Register base address
- ***WWDG_InitTypeDef WWDG_HandleTypeDef::Init***
  WWDG required parameters

72.2 WWDG Firmware driver API description

72.2.1 WWDG specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (T[6;0] downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls over from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.
- Once enabled the WWDG cannot be disabled except by a system reset.
- WWDGRST flag in RCC_CSR register informs when a WWDG reset has occurred (check available with __HAL_RCC_GET_FLAG(RCC_FLAG_WWDGRST)).

- The WWDG downcounter input clock is derived from the APB clock divided by a programmable prescaler.
- WWDG downcounter clock (Hz) = PCLK1 / (4096 * Prescaler)
- WWDG timeout (ms) = (1000 * (T[5;0] + 1)) / (WWDG downcounter clock) where T[5;0] are the lowest 6 bits of downcounter.
- WWDG Counter refresh is allowed between the following limits:
  - min time (ms) = (1000 * (T[5;0] - Window)) / (WWDG downcounter clock)
  - max time (ms) = (1000 * (T[5;0] - 0x40)) / (WWDG downcounter clock)
- Min-max timeout value @80 MHz(PCLK1): ~51.2 us / ~26.22 ms
- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. When the downcounter reaches the value 0x40, an EWI interrupt is generated and the corresponding interrupt service routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device. In some applications, the EWI interrupt can be used to manage a software system check and/or system recovery/graceful degradation, without generating a WWDG reset. In this case, the corresponding interrupt service routine (ISR) should reload the WWDG counter to avoid the WWDG reset, then trigger the required actions. Note:When the EWI interrupt cannot be served, e.g. due to a system lock in a higher priority task, the WWDG reset will eventually be generated.
- Debug mode: When the microcontroller enters debug mode (core halted), the WWDG counter either continues to work normally or stops, depending on DBG_WWDG_STOP configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_WWDG() and __HAL_DBGMCU_UNFREEZE_WWDG() macros

72.2.2 How to use this driver

- Enable WWDG APB1 clock using __HAL_RCC_WWDG_CLK_ENABLE().
- Set the WWDG prescaler, refresh window, counter value and Early Wakeup Interrupt mode using using HAL_WWDG_Init() function. This enables WWDG peripheral and the downcounter starts downcounting from given counter value. Init function can be called again to modify all watchdog parameters, however if EWI mode has been set once, it can't be clear until next reset.
- The application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset using HAL_WWDG_Refresh() function. This operation must occur only when the counter is lower than the window value already programmed.
- if Early Wakeup Interrupt mode is enable an interrupt is generated when the counter reaches 0x40. User can add his own code in weak function HAL_WWDG_EarlyWakeupCallback().

WWDG HAL driver macros list

Below the list of most used macros in WWDG HAL driver.

- __HAL_WWDG_GET_IT_SOURCE: Check the selected WWDG's interrupt source.
- __HAL_WWDG_GET_FLAG: Get the selected WWDG's flag status.
- __HAL_WWDG_CLEAR_FLAG: Clear the WWDG's pending flags.

72.2.3 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and start the WWDG according to the specified parameters in the WWDG_InitTypeDef of associated handle.
- Initialize the WWDG MSP.

This section contains the following APIs:

- ***HAL_WWDG_Init()***
- ***HAL_WWDG_MspInit()***

72.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the WWDG.
- Handle WWDG interrupt request and associated function callback.

This section contains the following APIs:

- ***HAL_WWDG_Refresh()***
- ***HAL_WWDG_IRQHandler()***
- ***HAL_WWDG_EarlyWakeupCallback()***

72.2.5 Detailed description of functions

HAL_WWDG_Init

Function name	**HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwwdg)**
Function description	Initialize the WWDG according to the specified.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **HAL:** status

HAL_WWDG_MspInit

Function name	**void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)**
Function description	Initialize the WWDG MSP.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **None:**
Notes	• When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_WWDG_Init function is called again to change parameters.

HAL_WWDG_Refresh

Function name	**HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg)**
Function description	Refresh the WWDG.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **HAL:** status

HAL_WWDG_IRQHandler

Function name	**void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)**
Function description	Handle WWDG interrupt request.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **None:**
Notes	• The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_WWDG_Init function with EWIMode set to WWDG_EWI_ENABLE. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

HAL_WWDG_EarlyWakeupCallback

Function name	**void HAL_WWDG_EarlyWakeupCallback (WWDG_HandleTypeDef * hwwdg)**
Function description	WWDG Early Wakeup callback.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **None:**

72.3 WWDG Firmware driver defines

72.3.1 WWDG

WWDG Early Wakeup Interrupt Mode

WWDG_EWI_DISABLE	EWI Disable
WWDG_EWI_ENABLE	EWI Enable

WWDG Exported Macros

__HAL_WWDG_ENABLE

Description:

- Enable the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_ENABLE_IT

Description:

- Enable the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to enable. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

__HAL_WWDG_GET_IT

Description:

- Check whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the it to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_IT

Description:

- Clear the WWDG interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

- Check whether the specified WWDG flag is set or not.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- The: new state of WWDG_FLAG (SET or

RESET).

__HAL_WWDG_CLEAR_FLAG	**Description:** • Clear the WWDG's pending flags. **Parameters:** • __HANDLE__: WWDG handle • __FLAG__: specifies the flag to clear. This parameter can be one of the following values: – WWDG_FLAG_EWIF: Early wakeup interrupt flag **Return value:** • None
__HAL_WWDG_GET_IT_SOURCE	**Description:** • Check whether the specified WWDG interrupt source is enabled or not. **Parameters:** • __HANDLE__: WWDG Handle. • __INTERRUPT__: specifies the WWDG interrupt source to check. This parameter can be one of the following values: – WWDG_IT_EWI: Early Wakeup Interrupt **Return value:** • state: of __INTERRUPT__ (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF	Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI	Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1	WWDG counter clock = (PCLK1/4096)/1
WWDG_PRESCALER_2	WWDG counter clock = (PCLK1/4096)/2
WWDG_PRESCALER_4	WWDG counter clock = (PCLK1/4096)/4
WWDG_PRESCALER_8	WWDG counter clock = (PCLK1/4096)/8

73 LL ADC Generic Driver

73.1 ADC Firmware driver registers structures

73.1.1 LL_ADC_CommonInitTypeDef

Data Fields

- ***uint32_t CommonClock***

Field Documentation

- ***uint32_t LL_ADC_CommonInitTypeDef::CommonClock***
  Set parameter common to several ADC: Clock source and prescaler. This parameter can be a value of ***ADC_LL_EC_COMMON_CLOCK_SOURCE***
  Note: On this STM32 serie, if ADC group injected is used, some clock ratio constraints between ADC clock and AHB clock must be respected. Refer to reference manual. This feature can be modified afterwards using unitary function **LL_ADC_SetCommonClock()**.

73.1.2 LL_ADC_InitTypeDef

Data Fields

- ***uint32_t Resolution***
- ***uint32_t DataAlignment***
- ***uint32_t LowPowerMode***

Field Documentation

- ***uint32_t LL_ADC_InitTypeDef::Resolution***
  Set ADC resolution. This parameter can be a value of ***ADC_LL_EC_RESOLUTION***This feature can be modified afterwards using unitary function **LL_ADC_SetResolution()**.
- ***uint32_t LL_ADC_InitTypeDef::DataAlignment***
  Set ADC conversion data alignment. This parameter can be a value of ***ADC_LL_EC_DATA_ALIGN***This feature can be modified afterwards using unitary function **LL_ADC_SetDataAlignment()**.
- ***uint32_t LL_ADC_InitTypeDef::LowPowerMode***
  Set ADC low power mode. This parameter can be a value of ***ADC_LL_EC_LP_MODE***This feature can be modified afterwards using unitary function **LL_ADC_SetLowPowerMode()**.

73.1.3 LL_ADC_REG_InitTypeDef

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t ContinuousMode***
- ***uint32_t DMATransfer***
- ***uint32_t Overrun***

Field Documentation

- ***uint32_t LL_ADC_REG_InitTypeDef::TriggerSource***
Set ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_REG_TRIGGER_SOURCE***
Note: On this STM32 serie, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_REG_SetTriggerEdge()**. This feature can be modified afterwards using unitary function **LL_ADC_REG_SetTriggerSource()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerLength***
Set ADC group regular sequencer length. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_SCAN_LENGTH***This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerLength()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerDiscont***
Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_DISCONT_MODE***
Note: This parameter has an effect only if group regular sequencer is enabled (scan length of 2 ranks or more). This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerDiscont()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::ContinuousMode***
Set ADC continuous conversion mode on ADC group regular, whether ADC conversions are performed in single mode (one conversion per trigger) or in continuous mode (after the first trigger, following conversions launched successively automatically). This parameter can be a value of ***ADC_LL_EC_REG_CONTINUOUS_MODE*** Note: It is not possible to enable both ADC group regular continuous mode and discontinuous mode.This feature can be modified afterwards using unitary function **LL_ADC_REG_SetContinuousMode()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::DMATransfer***
Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode. This parameter can be a value of ***ADC_LL_EC_REG_DMA_TRANSFER***This feature can be modified afterwards using unitary function **LL_ADC_REG_SetDMATransfer()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::Overrun***
Set ADC group regular behavior in case of overrun: data preserved or overwritten. This parameter can be a value of ***ADC_LL_EC_REG_OVR_DATA_BEHAVIOR***This feature can be modified afterwards using unitary function **LL_ADC_REG_SetOverrun()**.

73.1.4 LL_ADC_INJ_InitTypeDef

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t TrigAuto***

Field Documentation

- ***uint32_t LL_ADC_INJ_InitTypeDef::TriggerSource***
Set ADC group injected conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_INJ_TRIGGER_SOURCE***
Note: On this STM32 serie, setting trigger source to external trigger also set trigger

polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_INJ_SetTriggerEdge()**. This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTriggerSource()**.

- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerLength***
  Set ADC group injected sequencer length. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_SCAN_LENGTH***This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerLength()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerDiscont***
  Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_DISCONT_MODE***
  Note: This parameter has an effect only if group injected sequencer is enabled (scan length of 2 ranks or more). This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerDiscont()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::TrigAuto***
  Set ADC group injected conversion trigger: independent or from ADC group regular. This parameter can be a value of ***ADC_LL_EC_INJ_TRIG_AUTO*** Note: This parameter must be set to set to independent trigger if injected trigger source is set to an external trigger.This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTrigAuto()**.

73.2 ADC Firmware driver API description

73.2.1 Detailed description of functions

LL_ADC_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr (ADC_TypeDef * ADCx, uint32_t Register)**
Function description	Function to help to configure DMA transfer from ADC: retrieve the ADC register address from ADC instance and a list of ADC registers intended to be used (most commonly) with DMA transfer.
Parameters	• **ADCx:** ADC instance • **Register:** This parameter can be one of the following values: (1) Available on devices with several ADC instances. – LL_ADC_DMA_REG_REGULAR_DATA – LL_ADC_DMA_REG_REGULAR_DATA_MULTI (1)
Return values	• **ADC:** register address
Notes	• These ADC registers are data registers: when ADC conversion data is available in ADC data registers, ADC generates a DMA transfer request. • This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, LL_ADC_DMA_GetRegAddr(ADC1, LL_ADC_DMA_REG_REGULAR_DATA), (uint32_t)&< array or variable >, LL_DMA_DIRECTION_PERIPH_TO_MEMORY); • For devices with several ADC: in multimode, some devices use a different data register outside of ADC instance scope (common data register). This macro manages this register

	difference, only ADC instance has to be set as parameter.
Reference Manual to LL API cross reference:	• DR RDATA LL_ADC_DMA_GetRegAddr • CDR RDATA_MST LL_ADC_DMA_GetRegAddr • CDR RDATA_SLV LL_ADC_DMA_GetRegAddr

LL_ADC_SetCommonClock

Function name	**__STATIC_INLINE void LL_ADC_SetCommonClock (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t CommonClock)**
Function description	Set parameter common to several ADC: Clock source and prescaler.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE()) • **CommonClock:** This parameter can be one of the following values: – LL_ADC_CLOCK_SYNC_PCLK_DIV1 – LL_ADC_CLOCK_SYNC_PCLK_DIV2 – LL_ADC_CLOCK_SYNC_PCLK_DIV4 – LL_ADC_CLOCK_ASYNC_DIV1 – LL_ADC_CLOCK_ASYNC_DIV2 – LL_ADC_CLOCK_ASYNC_DIV4 – LL_ADC_CLOCK_ASYNC_DIV6 – LL_ADC_CLOCK_ASYNC_DIV8 – LL_ADC_CLOCK_ASYNC_DIV10 – LL_ADC_CLOCK_ASYNC_DIV12 – LL_ADC_CLOCK_ASYNC_DIV16 – LL_ADC_CLOCK_ASYNC_DIV32 – LL_ADC_CLOCK_ASYNC_DIV64 – LL_ADC_CLOCK_ASYNC_DIV128 – LL_ADC_CLOCK_ASYNC_DIV256
Return values	• **None:**
Notes	• On this STM32 serie, if ADC group injected is used, some clock ratio constraints between ADC clock and AHB clock must be respected. Refer to reference manual. • On this STM32 serie, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().
Reference Manual to LL API cross reference:	• CCR CKMODE LL_ADC_SetCommonClock • CCR PRESC LL_ADC_SetCommonClock

LL_ADC_GetCommonClock

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetCommonClock (ADC_Common_TypeDef * ADCxy_COMMON)**

Function description	Get parameter common to several ADC: Clock source and prescaler.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **Returned:** value can be one of the following values: – LL_ADC_CLOCK_SYNC_PCLK_DIV1 – LL_ADC_CLOCK_SYNC_PCLK_DIV2 – LL_ADC_CLOCK_SYNC_PCLK_DIV4 – LL_ADC_CLOCK_ASYNC_DIV1 – LL_ADC_CLOCK_ASYNC_DIV2 – LL_ADC_CLOCK_ASYNC_DIV4 – LL_ADC_CLOCK_ASYNC_DIV6 – LL_ADC_CLOCK_ASYNC_DIV8 – LL_ADC_CLOCK_ASYNC_DIV10 – LL_ADC_CLOCK_ASYNC_DIV12 – LL_ADC_CLOCK_ASYNC_DIV16 – LL_ADC_CLOCK_ASYNC_DIV32 – LL_ADC_CLOCK_ASYNC_DIV64 – LL_ADC_CLOCK_ASYNC_DIV128 – LL_ADC_CLOCK_ASYNC_DIV256
Reference Manual to LL API cross reference:	• CCR CKMODE LL_ADC_GetCommonClock • CCR PRESC LL_ADC_GetCommonClock

LL_ADC_SetCommonPathInternalCh

Function name	**__STATIC_INLINE void LL_ADC_SetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)**
Function description	Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE()) • **PathInternal:** This parameter can be a combination of the following values: – LL_ADC_PATH_INTERNAL_NONE – LL_ADC_PATH_INTERNAL_VREFINT – LL_ADC_PATH_INTERNAL_TEMPSENSOR – LL_ADC_PATH_INTERNAL_VBAT
Return values	• **None:**
Notes	• One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR) • Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal

	LL_ADC_DELAY_VREFINT_STAB_US. Refer to literal LL_ADC_DELAY_TEMPSENSOR_STAB_US. • ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet. • On this STM32 serie, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().
Reference Manual to LL API cross reference:	• CCR VREFEN LL_ADC_SetCommonPathInternalCh • CCR TSEN LL_ADC_SetCommonPathInternalCh • CCR VBATEN LL_ADC_SetCommonPathInternalCh

LL_ADC_GetCommonPathInternalCh

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **Returned:** value can be a combination of the following values: – LL_ADC_PATH_INTERNAL_NONE – LL_ADC_PATH_INTERNAL_VREFINT – LL_ADC_PATH_INTERNAL_TEMPSENSOR – LL_ADC_PATH_INTERNAL_VBAT
Notes	• One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
Reference Manual to LL API cross reference:	• CCR VREFEN LL_ADC_GetCommonPathInternalCh • CCR TSEN LL_ADC_GetCommonPathInternalCh • CCR VBATEN LL_ADC_GetCommonPathInternalCh

LL_ADC_SetCalibrationFactor

Function name	**__STATIC_INLINE void LL_ADC_SetCalibrationFactor (ADC_TypeDef * ADCx, uint32_t SingleDiff, uint32_t CalibrationFactor)**
Function description	Set ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).
Parameters	• **ADCx:** ADC instance • **SingleDiff:** This parameter can be one of the following values: – LL_ADC_SINGLE_ENDED – LL_ADC_DIFFERENTIAL_ENDED

	– LL_ADC_BOTH_SINGLE_DIFF_ENDED • **CalibrationFactor:** Value between Min_Data=0x00 and Max_Data=0x7F
Return values	• **None:**
Notes	• This function is intended to set calibration parameters without having to perform a new calibration using LL_ADC_StartCalibration(). • For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration factor must be specified for each of these differential modes, if used afterwards and if the application requires their calibration). • In case of setting calibration factors of both modes single ended and differential (parameter LL_ADC_BOTH_SINGLE_DIFF_ENDED): both calibration factors must be concatenated. To perform this processing, use helper macro __LL_ADC_CALIB_FACTOR_SINGLE_DIFF(). • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled, without calibration on going, without conversion on going on group regular.
Reference Manual to LL API cross reference:	• CALFACT CALFACT_S LL_ADC_SetCalibrationFactor • CALFACT CALFACT_D LL_ADC_SetCalibrationFactor

LL_ADC_GetCalibrationFactor

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetCalibrationFactor (ADC_TypeDef * ADCx, uint32_t SingleDiff)**
Function description	Get ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).
Parameters	• **ADCx:** ADC instance • **SingleDiff:** This parameter can be one of the following values: – LL_ADC_SINGLE_ENDED – LL_ADC_DIFFERENTIAL_ENDED
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x7F
Notes	• Calibration factors are set by hardware after performing a calibration run using function LL_ADC_StartCalibration(). • For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes
Reference Manual to LL API cross reference:	• CALFACT CALFACT_S LL_ADC_GetCalibrationFactor • CALFACT CALFACT_D LL_ADC_GetCalibrationFactor

LL_ADC_SetResolution

Function name	**__STATIC_INLINE void LL_ADC_SetResolution (ADC_TypeDef * ADCx, uint32_t Resolution)**

Function description	Set ADC resolution.
Parameters	• **ADCx:** ADC instance • **Resolution:** This parameter can be one of the following values: – LL_ADC_RESOLUTION_12B – LL_ADC_RESOLUTION_10B – LL_ADC_RESOLUTION_8B – LL_ADC_RESOLUTION_6B
Return values	• **None:**
Notes	• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• CFGR RES LL_ADC_SetResolution

LL_ADC_GetResolution

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetResolution (ADC_TypeDef * ADCx)**
Function description	Get ADC resolution.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_RESOLUTION_12B – LL_ADC_RESOLUTION_10B – LL_ADC_RESOLUTION_8B – LL_ADC_RESOLUTION_6B
Reference Manual to LL API cross reference:	• CFGR RES LL_ADC_GetResolution

LL_ADC_SetDataAlignment

Function name	**__STATIC_INLINE void LL_ADC_SetDataAlignment (ADC_TypeDef * ADCx, uint32_t DataAlignment)**
Function description	Set ADC conversion data alignment.
Parameters	• **ADCx:** ADC instance • **DataAlignment:** This parameter can be one of the following values: – LL_ADC_DATA_ALIGN_RIGHT – LL_ADC_DATA_ALIGN_LEFT
Return values	• **None:**
Notes	• Refer to reference manual for alignments formats dependencies to ADC resolutions. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:	• CFGR ALIGN LL_ADC_SetDataAlignment

LL_ADC_GetDataAlignment

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetDataAlignment (ADC_TypeDef * ADCx)**
Function description	Get ADC conversion data alignment.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_DATA_ALIGN_RIGHT – LL_ADC_DATA_ALIGN_LEFT
Notes	• Refer to reference manual for alignments formats dependencies to ADC resolutions.
Reference Manual to LL API cross reference:	• CFGR ALIGN LL_ADC_GetDataAlignment

LL_ADC_SetLowPowerMode

Function name	**__STATIC_INLINE void LL_ADC_SetLowPowerMode (ADC_TypeDef * ADCx, uint32_t LowPowerMode)**
Function description	Set ADC low power mode.
Parameters	• **ADCx:** ADC instance • **LowPowerMode:** This parameter can be one of the following values: – LL_ADC_LP_MODE_NONE – LL_ADC_LP_AUTOWAIT
Return values	• **None:**
Notes	• Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: Do not use with interruption or DMA since these modes have to clear immediately the EOC flag to free the IRQ vector sequencer.Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter

	LL_ADC_LP_MODE_AUTOOFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait". • With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• CFGR AUTDLY LL_ADC_SetLowPowerMode

LL_ADC_GetLowPowerMode

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetLowPowerMode (ADC_TypeDef * ADCx)**
Function description	Get ADC low power mode:
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_LP_MODE_NONE – LL_ADC_LP_AUTOWAIT
Notes	• Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: Do not use with interruption or DMA since these modes have to clear immediately the EOC flag to free the IRQ vector sequencer.Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_MODE_AUTOOFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".

- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.

Reference Manual to LL API cross reference:
- CFGR AUTDLY LL_ADC_GetLowPowerMode

LL_ADC_SetOffset

Function name
__STATIC_INLINE void LL_ADC_SetOffset (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t Channel, uint32_t OffsetLevel)

Function description
Set ADC selected offset number 1, 2, 3 or 4.

Parameters
- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **Channel:** This parameter can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (7)
  - LL_ADC_CHANNEL_2 (7)
  - LL_ADC_CHANNEL_3 (7)
  - LL_ADC_CHANNEL_4 (7)
  - LL_ADC_CHANNEL_5 (7)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (4)
  - LL_ADC_CHANNEL_VBAT (4)
  - LL_ADC_CHANNEL_DAC1CH1 (5)
  - LL_ADC_CHANNEL_DAC1CH2 (5)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)

- – LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)
- (2) On STM32L4, parameter available only on ADC instance: ADC2.
- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3.
- (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.
- (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion rate up to 4.21 Ms/s)).
- **OffsetLevel:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values	• **None:**
Notes	• This function set the 2 items of offset configuration: ADC channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)Offset level (offset to be subtracted from the raw converted data). • Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0. • This function enables the offset, by default. It can be forced to disable state using function LL_ADC_SetOffsetState(). • If a channel is mapped on several offsets numbers, only the offset with the lowest value is considered for the subtraction. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected. • On STM32L4, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN1..5).
Reference Manual to LL API cross reference:	• OFR1 OFFSET1_CH LL_ADC_SetOffset • OFR1 OFFSET1 LL_ADC_SetOffset • OFR1 OFFSET1_EN LL_ADC_SetOffset • OFR2 OFFSET2_CH LL_ADC_SetOffset • OFR2 OFFSET2 LL_ADC_SetOffset • OFR2 OFFSET2_EN LL_ADC_SetOffset • OFR3 OFFSET3_CH LL_ADC_SetOffset • OFR3 OFFSET3 LL_ADC_SetOffset • OFR3 OFFSET3_EN LL_ADC_SetOffset • OFR4 OFFSET4_CH LL_ADC_SetOffset • OFR4 OFFSET4 LL_ADC_SetOffset • OFR4 OFFSET4_EN LL_ADC_SetOffset

LL_ADC_GetOffsetChannel

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOffsetChannel (ADC_TypeDef * ADCx, uint32_t Offsety)**

Function description	Get for the ADC selected offset number 1, 2, 3 or 4: Channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)
Parameters	• **ADCx:** ADC instance • **Offsety:** This parameter can be one of the following values: – LL_ADC_OFFSET_1 – LL_ADC_OFFSET_2 – LL_ADC_OFFSET_3 – LL_ADC_OFFSET_4
Return values	• **Returned:** value can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1. – LL_ADC_CHANNEL_0 – LL_ADC_CHANNEL_1 (7) – LL_ADC_CHANNEL_2 (7) – LL_ADC_CHANNEL_3 (7) – LL_ADC_CHANNEL_4 (7) – LL_ADC_CHANNEL_5 (7) – LL_ADC_CHANNEL_6 – LL_ADC_CHANNEL_7 – LL_ADC_CHANNEL_8 – LL_ADC_CHANNEL_9 – LL_ADC_CHANNEL_10 – LL_ADC_CHANNEL_11 – LL_ADC_CHANNEL_12 – LL_ADC_CHANNEL_13 – LL_ADC_CHANNEL_14 – LL_ADC_CHANNEL_15 – LL_ADC_CHANNEL_16 – LL_ADC_CHANNEL_17 – LL_ADC_CHANNEL_18 – LL_ADC_CHANNEL_VREFINT (1) – LL_ADC_CHANNEL_TEMPSENSOR (4) – LL_ADC_CHANNEL_VBAT (4) – LL_ADC_CHANNEL_DAC1CH1 (5) – LL_ADC_CHANNEL_DAC1CH2 (5) – LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6) – LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6) – LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6) – LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6) • (2) On STM32L4, parameter available only on ADC instance: ADC2. • (3) On STM32L4, parameter available only on ADC instance: ADC3. • (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3. • (5) On STM32L4, parameter available on devices with only 1 ADC instance. • (6) On STM32L4, parameter available on devices with several ADC instances. • (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion

	rate up to 4.21 Ms/s)). • (1, 2, 3, 4) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().
Notes	• Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). • On STM32L4, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN1..5).
Reference Manual to LL API cross reference:	• OFR1 OFFSET1_CH LL_ADC_GetOffsetChannel • OFR2 OFFSET2_CH LL_ADC_GetOffsetChannel • OFR3 OFFSET3_CH LL_ADC_GetOffsetChannel • OFR4 OFFSET4_CH LL_ADC_GetOffsetChannel

LL_ADC_GetOffsetLevel

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOffsetLevel (ADC_TypeDef * ADCx, uint32_t Offsety)**
Function description	Get for the ADC selected offset number 1, 2, 3 or 4: Offset level (offset to be subtracted from the raw converted data).
Parameters	• **ADCx:** ADC instance • **Offsety:** This parameter can be one of the following values: – LL_ADC_OFFSET_1 – LL_ADC_OFFSET_2 – LL_ADC_OFFSET_3 – LL_ADC_OFFSET_4
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0xFFF
Notes	• Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.
Reference Manual to LL API cross reference:	• OFR1 OFFSET1 LL_ADC_GetOffsetLevel • OFR2 OFFSET2 LL_ADC_GetOffsetLevel • OFR3 OFFSET3 LL_ADC_GetOffsetLevel • OFR4 OFFSET4 LL_ADC_GetOffsetLevel

LL_ADC_SetOffsetState

Function name	**__STATIC_INLINE void LL_ADC_SetOffsetState (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetState)**
Function description	Set for the ADC selected offset number 1, 2, 3 or 4: force offset state disable or enable without modifying offset channel or offset

value.

Parameters	• **ADCx:** ADC instance • **Offsety:** This parameter can be one of the following values: – LL_ADC_OFFSET_1 – LL_ADC_OFFSET_2 – LL_ADC_OFFSET_3 – LL_ADC_OFFSET_4 • **OffsetState:** This parameter can be one of the following values: – LL_ADC_OFFSET_DISABLE – LL_ADC_OFFSET_ENABLE
Return values	• **None:**
Notes	• This function should be needed only in case of offset to be enabled-disabled dynamically, and should not be needed in other cases: function LL_ADC_SetOffset() automatically enables the offset. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• OFR1 OFFSET1_EN LL_ADC_SetOffsetState • OFR2 OFFSET2_EN LL_ADC_SetOffsetState • OFR3 OFFSET3_EN LL_ADC_SetOffsetState • OFR4 OFFSET4_EN LL_ADC_SetOffsetState

LL_ADC_GetOffsetState

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOffsetState (ADC_TypeDef * ADCx, uint32_t Offsety)**
Function description	Get for the ADC selected offset number 1, 2, 3 or 4: offset state disabled or enabled.
Parameters	• **ADCx:** ADC instance • **Offsety:** This parameter can be one of the following values: – LL_ADC_OFFSET_1 – LL_ADC_OFFSET_2 – LL_ADC_OFFSET_3 – LL_ADC_OFFSET_4
Return values	• **Returned:** value can be one of the following values: – LL_ADC_OFFSET_DISABLE – LL_ADC_OFFSET_ENABLE
Reference Manual to LL API cross reference:	• OFR1 OFFSET1_EN LL_ADC_GetOffsetState • OFR2 OFFSET2_EN LL_ADC_GetOffsetState • OFR3 OFFSET3_EN LL_ADC_GetOffsetState • OFR4 OFFSET4_EN LL_ADC_GetOffsetState

LL_ADC_SetSamplingTimeCommonConfig

Function name	**__STATIC_INLINE void LL_ADC_SetSamplingTimeCommonConfig (ADC_TypeDef * ADCx, uint32_t SamplingTimeCommonConfig)**

Function description	Set ADC sampling time common configuration impacting settings of sampling time channel wise.
Parameters	• **ADCx:** ADC instance • **SamplingTimeCommonConfig:** This parameter can be one of the following values: – LL_ADC_SAMPLINGTIME_COMMON_DEFAULT – LL_ADC_SAMPLINGTIME_COMMON_3C5_REPL_2C5
Return values	• **None:**
Notes	• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• SMPR1 SMPPLUS LL_ADC_SetSamplingTimeCommonConfig

LL_ADC_GetSamplingTimeCommonConfig

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetSamplingTimeCommonConfig (ADC_TypeDef * ADCx)**
Function description	Get ADC sampling time common configuration impacting settings of sampling time channel wise.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_SAMPLINGTIME_COMMON_DEFAULT – LL_ADC_SAMPLINGTIME_COMMON_3C5_REPL_2C5
Reference Manual to LL API cross reference:	• SMPR1 SMPPLUS LL_ADC_GetSamplingTimeCommonConfig

LL_ADC_REG_SetTriggerSource

Function name	**__STATIC_INLINE void LL_ADC_REG_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)**
Function description	Set ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).
Parameters	• **ADCx:** ADC instance • **TriggerSource:** This parameter can be one of the following values: – LL_ADC_REG_TRIG_SOFTWARE – LL_ADC_REG_TRIG_EXT_TIM1_TRGO – LL_ADC_REG_TRIG_EXT_TIM1_TRGO2 – LL_ADC_REG_TRIG_EXT_TIM1_CH1 – LL_ADC_REG_TRIG_EXT_TIM1_CH2 – LL_ADC_REG_TRIG_EXT_TIM1_CH3 – LL_ADC_REG_TRIG_EXT_TIM2_TRGO – LL_ADC_REG_TRIG_EXT_TIM2_CH2 – LL_ADC_REG_TRIG_EXT_TIM3_TRGO – LL_ADC_REG_TRIG_EXT_TIM3_CH4

  - LL_ADC_REG_TRIG_EXT_TIM4_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4
  - LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM15_TRGO
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11

Return values	• **None:**
Notes	• On this STM32 serie, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_REG_SetTriggerEdge(). • Availability of parameters of trigger sources from timer depends on timers availability on the selected device. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	• CFGR EXTSEL LL_ADC_REG_SetTriggerSource • CFGR EXTEN LL_ADC_REG_SetTriggerSource

LL_ADC_REG_GetTriggerSource

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerSource (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_TRIG_SOFTWARE – LL_ADC_REG_TRIG_EXT_TIM1_TRGO – LL_ADC_REG_TRIG_EXT_TIM1_TRGO2 – LL_ADC_REG_TRIG_EXT_TIM1_CH1 – LL_ADC_REG_TRIG_EXT_TIM1_CH2 – LL_ADC_REG_TRIG_EXT_TIM1_CH3 – LL_ADC_REG_TRIG_EXT_TIM2_TRGO – LL_ADC_REG_TRIG_EXT_TIM2_CH2 – LL_ADC_REG_TRIG_EXT_TIM3_TRGO – LL_ADC_REG_TRIG_EXT_TIM3_CH4 – LL_ADC_REG_TRIG_EXT_TIM4_TRGO – LL_ADC_REG_TRIG_EXT_TIM4_CH4 – LL_ADC_REG_TRIG_EXT_TIM6_TRGO – LL_ADC_REG_TRIG_EXT_TIM8_TRGO – LL_ADC_REG_TRIG_EXT_TIM8_TRGO2 – LL_ADC_REG_TRIG_EXT_TIM15_TRGO – LL_ADC_REG_TRIG_EXT_EXTI_LINE11
Notes	• To determine whether group regular trigger source is internal (SW start) or external, without detail of which peripheral is

	selected as external trigger, (equivalent to "if(LL_ADC_REG_GetTriggerSource(ADC1) == LL_ADC_REG_TRIG_SOFTWARE)") use function LL_ADC_REG_IsTriggerSourceSWStart. • Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
Reference Manual to LL API cross reference:	• CFGR EXTSEL LL_ADC_REG_GetTriggerSource • CFGR EXTEN LL_ADC_REG_GetTriggerSource

LL_ADC_REG_IsTriggerSourceSWStart

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_IsTriggerSourceSWStart (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion trigger source internal (SW start) or external.
Parameters	• **ADCx:** ADC instance
Return values	• **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.
Notes	• In case of group regular trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_REG_GetTriggerSource().
Reference Manual to LL API cross reference:	• CFGR EXTEN LL_ADC_REG_IsTriggerSourceSWStart

LL_ADC_REG_SetTriggerEdge

Function name	**__STATIC_INLINE void LL_ADC_REG_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)**
Function description	Set ADC group regular conversion trigger polarity.
Parameters	• **ADCx:** ADC instance • **ExternalTriggerEdge:** This parameter can be one of the following values: – LL_ADC_REG_TRIG_EXT_RISING – LL_ADC_REG_TRIG_EXT_FALLING – LL_ADC_REG_TRIG_EXT_RISINGFALLING
Return values	• **None:**
Notes	• Applicable only for trigger source set to external trigger. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	• CFGR EXTEN LL_ADC_REG_SetTriggerEdge

LL_ADC_REG_GetTriggerEdge

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerEdge (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion trigger polarity.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_TRIG_EXT_RISING – LL_ADC_REG_TRIG_EXT_FALLING – LL_ADC_REG_TRIG_EXT_RISINGFALLING
Notes	• Applicable only for trigger source set to external trigger.
Reference Manual to LL API cross reference:	• CFGR EXTEN LL_ADC_REG_GetTriggerEdge

LL_ADC_REG_SetSequencerLength

Function name	**__STATIC_INLINE void LL_ADC_REG_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)**
Function description	Set ADC group regular sequencer length and scan direction.
Parameters	• **ADCx:** ADC instance • **SequencerNbRanks:** This parameter can be one of the following values: – LL_ADC_REG_SEQ_SCAN_DISABLE – LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS
Return values	• **None:**
Notes	• Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function

	"LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function performs configuration of: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()". • Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	• SQR1 L LL_ADC_REG_SetSequencerLength

LL_ADC_REG_GetSequencerLength

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerLength (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular sequencer length and scan direction.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_SEQ_SCAN_DISABLE – LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS – LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS
Notes	• Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function retrieves: Sequence length:

	Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function retrieves: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()". • Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
Reference Manual to LL API cross reference:	• SQR1 L LL_ADC_REG_GetSequencerLength

LL_ADC_REG_SetSequencerDiscont

Function name	**__STATIC_INLINE void LL_ADC_REG_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)**
Function description	Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.
Parameters	• **ADCx:** ADC instance • **SeqDiscont:** This parameter can be one of the following values: – LL_ADC_REG_SEQ_DISCONT_DISABLE – LL_ADC_REG_SEQ_DISCONT_1RANK – LL_ADC_REG_SEQ_DISCONT_2RANKS – LL_ADC_REG_SEQ_DISCONT_3RANKS – LL_ADC_REG_SEQ_DISCONT_4RANKS – LL_ADC_REG_SEQ_DISCONT_5RANKS – LL_ADC_REG_SEQ_DISCONT_6RANKS – LL_ADC_REG_SEQ_DISCONT_7RANKS – LL_ADC_REG_SEQ_DISCONT_8RANKS
Return values	• **None:**
Notes	• It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode. • It is not possible to enable both ADC auto-injected mode and ADC group regular sequencer discontinuous mode. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross	• CFGR DISCEN LL_ADC_REG_SetSequencerDiscont • CFGR DISCNUM LL_ADC_REG_SetSequencerDiscont

reference:

LL_ADC_REG_GetSequencerDiscont

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerDiscont (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_SEQ_DISCONT_DISABLE – LL_ADC_REG_SEQ_DISCONT_1RANK – LL_ADC_REG_SEQ_DISCONT_2RANKS – LL_ADC_REG_SEQ_DISCONT_3RANKS – LL_ADC_REG_SEQ_DISCONT_4RANKS – LL_ADC_REG_SEQ_DISCONT_5RANKS – LL_ADC_REG_SEQ_DISCONT_6RANKS – LL_ADC_REG_SEQ_DISCONT_7RANKS – LL_ADC_REG_SEQ_DISCONT_8RANKS
Reference Manual to LL API cross reference:	• CFGR DISCEN LL_ADC_REG_GetSequencerDiscont • CFGR DISCNUM LL_ADC_REG_GetSequencerDiscont

LL_ADC_REG_SetSequencerRanks

Function name	**__STATIC_INLINE void LL_ADC_REG_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)**
Function description	Set ADC group regular sequence: channel on the selected scan sequence rank.
Parameters	• **ADCx:** ADC instance • **Rank:** This parameter can be one of the following values: – LL_ADC_REG_RANK_1 – LL_ADC_REG_RANK_2 – LL_ADC_REG_RANK_3 – LL_ADC_REG_RANK_4 – LL_ADC_REG_RANK_5 – LL_ADC_REG_RANK_6 – LL_ADC_REG_RANK_7 – LL_ADC_REG_RANK_8 – LL_ADC_REG_RANK_9 – LL_ADC_REG_RANK_10 – LL_ADC_REG_RANK_11 – LL_ADC_REG_RANK_12 – LL_ADC_REG_RANK_13 – LL_ADC_REG_RANK_14 – LL_ADC_REG_RANK_15 – LL_ADC_REG_RANK_16 • **Channel:** This parameter can be one of the following values: (1) On STM32L4, parameter available only on ADC instance:

ADC1.

- LL_ADC_CHANNEL_0
- LL_ADC_CHANNEL_1 (7)
- LL_ADC_CHANNEL_2 (7)
- LL_ADC_CHANNEL_3 (7)
- LL_ADC_CHANNEL_4 (7)
- LL_ADC_CHANNEL_5 (7)
- LL_ADC_CHANNEL_6
- LL_ADC_CHANNEL_7
- LL_ADC_CHANNEL_8
- LL_ADC_CHANNEL_9
- LL_ADC_CHANNEL_10
- LL_ADC_CHANNEL_11
- LL_ADC_CHANNEL_12
- LL_ADC_CHANNEL_13
- LL_ADC_CHANNEL_14
- LL_ADC_CHANNEL_15
- LL_ADC_CHANNEL_16
- LL_ADC_CHANNEL_17
- LL_ADC_CHANNEL_18
- LL_ADC_CHANNEL_VREFINT (1)
- LL_ADC_CHANNEL_TEMPSENSOR (4)
- LL_ADC_CHANNEL_VBAT (4)
- LL_ADC_CHANNEL_DAC1CH1 (5)
- LL_ADC_CHANNEL_DAC1CH2 (5)
- LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
- LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
- LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
- LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

- (2) On STM32L4, parameter available only on ADC instance: ADC2.
- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3.
- (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.
- (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion rate up to 4.21 Ms/s)).

Return values

- **None:**

Notes

- This function performs configuration of: Channels ordering into each rank of scan sequence: whatever channel can be placed into whatever rank.
- On this STM32 serie, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may

	not be available. Refer to device datasheet for channels availability. • On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh(). • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	• SQR1 SQ1 LL_ADC_REG_SetSequencerRanks • SQR1 SQ2 LL_ADC_REG_SetSequencerRanks • SQR1 SQ3 LL_ADC_REG_SetSequencerRanks • SQR1 SQ4 LL_ADC_REG_SetSequencerRanks • SQR2 SQ5 LL_ADC_REG_SetSequencerRanks • SQR2 SQ6 LL_ADC_REG_SetSequencerRanks • SQR2 SQ7 LL_ADC_REG_SetSequencerRanks • SQR2 SQ8 LL_ADC_REG_SetSequencerRanks • SQR2 SQ9 LL_ADC_REG_SetSequencerRanks • SQR3 SQ10 LL_ADC_REG_SetSequencerRanks • SQR3 SQ11 LL_ADC_REG_SetSequencerRanks • SQR3 SQ12 LL_ADC_REG_SetSequencerRanks • SQR3 SQ13 LL_ADC_REG_SetSequencerRanks • SQR3 SQ14 LL_ADC_REG_SetSequencerRanks • SQR4 SQ15 LL_ADC_REG_SetSequencerRanks • SQR4 SQ16 LL_ADC_REG_SetSequencerRanks

LL_ADC_REG_GetSequencerRanks

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank)**
Function description	Get ADC group regular sequence: channel on the selected scan sequence rank.
Parameters	• **ADCx:** ADC instance • **Rank:** This parameter can be one of the following values: – LL_ADC_REG_RANK_1 – LL_ADC_REG_RANK_2 – LL_ADC_REG_RANK_3 – LL_ADC_REG_RANK_4 – LL_ADC_REG_RANK_5 – LL_ADC_REG_RANK_6 – LL_ADC_REG_RANK_7 – LL_ADC_REG_RANK_8 – LL_ADC_REG_RANK_9 – LL_ADC_REG_RANK_10 – LL_ADC_REG_RANK_11 – LL_ADC_REG_RANK_12 – LL_ADC_REG_RANK_13 – LL_ADC_REG_RANK_14 – LL_ADC_REG_RANK_15 – LL_ADC_REG_RANK_16

Return values	• **Returned:** value can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1. – LL_ADC_CHANNEL_0 – LL_ADC_CHANNEL_1 (7) – LL_ADC_CHANNEL_2 (7) – LL_ADC_CHANNEL_3 (7) – LL_ADC_CHANNEL_4 (7) – LL_ADC_CHANNEL_5 (7) – LL_ADC_CHANNEL_6 – LL_ADC_CHANNEL_7 – LL_ADC_CHANNEL_8 – LL_ADC_CHANNEL_9 – LL_ADC_CHANNEL_10 – LL_ADC_CHANNEL_11 – LL_ADC_CHANNEL_12 – LL_ADC_CHANNEL_13 – LL_ADC_CHANNEL_14 – LL_ADC_CHANNEL_15 – LL_ADC_CHANNEL_16 – LL_ADC_CHANNEL_17 – LL_ADC_CHANNEL_18 – LL_ADC_CHANNEL_VREFINT (1) – LL_ADC_CHANNEL_TEMPSENSOR (4) – LL_ADC_CHANNEL_VBAT (4) – LL_ADC_CHANNEL_DAC1CH1 (5) – LL_ADC_CHANNEL_DAC1CH2 (5) – LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6) – LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6) – LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6) – LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6) • (2) On STM32L4, parameter available only on ADC instance: ADC2. • (3) On STM32L4, parameter available only on ADC instance: ADC3. • (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3. • (5) On STM32L4, parameter available on devices with only 1 ADC instance. • (6) On STM32L4, parameter available on devices with several ADC instances. • (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion rate up to 4.21 Ms/s)). • (1, 2, 3, 4) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().
Notes	• On this STM32 serie, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength(). • Depending on devices and packages, some channels may

	not be available. Refer to device datasheet for channels availability. • Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().
Reference Manual to LL API cross reference:	• SQR1 SQ1 LL_ADC_REG_GetSequencerRanks • SQR1 SQ2 LL_ADC_REG_GetSequencerRanks • SQR1 SQ3 LL_ADC_REG_GetSequencerRanks • SQR1 SQ4 LL_ADC_REG_GetSequencerRanks • SQR2 SQ5 LL_ADC_REG_GetSequencerRanks • SQR2 SQ6 LL_ADC_REG_GetSequencerRanks • SQR2 SQ7 LL_ADC_REG_GetSequencerRanks • SQR2 SQ8 LL_ADC_REG_GetSequencerRanks • SQR2 SQ9 LL_ADC_REG_GetSequencerRanks • SQR3 SQ10 LL_ADC_REG_GetSequencerRanks • SQR3 SQ11 LL_ADC_REG_GetSequencerRanks • SQR3 SQ12 LL_ADC_REG_GetSequencerRanks • SQR3 SQ13 LL_ADC_REG_GetSequencerRanks • SQR3 SQ14 LL_ADC_REG_GetSequencerRanks • SQR4 SQ15 LL_ADC_REG_GetSequencerRanks • SQR4 SQ16 LL_ADC_REG_GetSequencerRanks

LL_ADC_REG_SetContinuousMode

Function name	**__STATIC_INLINE void LL_ADC_REG_SetContinuousMode (ADC_TypeDef * ADCx, uint32_t Continuous)**
Function description	Set ADC continuous conversion mode on ADC group regular.
Parameters	• **ADCx:** ADC instance • **Continuous:** This parameter can be one of the following values: – LL_ADC_REG_CONV_SINGLE – LL_ADC_REG_CONV_CONTINUOUS
Return values	• **None:**
Notes	• Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically. • It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross	• CFGR CONT LL_ADC_REG_SetContinuousMode

reference:

LL_ADC_REG_GetContinuousMode

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetContinuousMode (ADC_TypeDef * ADCx)**
Function description	Get ADC continuous conversion mode on ADC group regular.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_CONV_SINGLE – LL_ADC_REG_CONV_CONTINUOUS
Notes	• Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
Reference Manual to LL API cross reference:	• CFGR CONT LL_ADC_REG_GetContinuousMode

LL_ADC_REG_SetDMATransfer

Function name	**__STATIC_INLINE void LL_ADC_REG_SetDMATransfer (ADC_TypeDef * ADCx, uint32_t DMATransfer)**
Function description	Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.
Parameters	• **ADCx:** ADC instance • **DMATransfer:** This parameter can be one of the following values: – LL_ADC_REG_DMA_TRANSFER_NONE – LL_ADC_REG_DMA_TRANSFER_LIMITED – LL_ADC_REG_DMA_TRANSFER_UNLIMITED
Return values	• **None:**
Notes	• If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular. • If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled). • For devices with several ADC instances: ADC multimode DMA settings are available using function LL_ADC_SetMultiDMATransfer(). • To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().

	• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• CFGR DMAEN LL_ADC_REG_SetDMATransfer • CFGR DMACFG LL_ADC_REG_SetDMATransfer

LL_ADC_REG_GetDMATransfer

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetDMATransfer (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_DMA_TRANSFER_NONE – LL_ADC_REG_DMA_TRANSFER_LIMITED – LL_ADC_REG_DMA_TRANSFER_UNLIMITED
Notes	• If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular. • If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled). • For devices with several ADC instances: ADC multimode DMA settings are available using function LL_ADC_GetMultiDMATransfer(). • To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().
Reference Manual to LL API cross reference:	• CFGR DMAEN LL_ADC_REG_GetDMATransfer • CFGR DMACFG LL_ADC_REG_GetDMATransfer

LL_ADC_REG_SetDFSDMTransfer

Function name	**__STATIC_INLINE void LL_ADC_REG_SetDFSDMTransfer (ADC_TypeDef * ADCx, uint32_t DFSDMTransfer)**
Function description	Set ADC group regular conversion data transfer to DFSDM.
Parameters	• **ADCx:** ADC instance • **DFSDMTransfer:** This parameter can be one of the following values: – LL_ADC_REG_DFSDM_TRANSFER_NONE – LL_ADC_REG_DFSDM_TRANSFER_ENABLE

Return values	• **None:**
Notes	• DFSDM transfer cannot be used if DMA transfer is enabled. • To configure DFSDM source address (peripheral address), use the same function as for DMA transfer: function LL_ADC_DMA_GetRegAddr(). • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• CFGR DFSDMCFG LL_ADC_REG_GetDFSDMTransfer

LL_ADC_REG_GetDFSDMTransfer

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetDFSDMTransfer (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data transfer to DFSDM.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_DFSDM_TRANSFER_NONE – LL_ADC_REG_DFSDM_TRANSFER_ENABLE
Reference Manual to LL API cross reference:	• CFGR DFSDMCFG LL_ADC_REG_GetDFSDMTransfer

LL_ADC_REG_SetOverrun

Function name	**__STATIC_INLINE void LL_ADC_REG_SetOverrun (ADC_TypeDef * ADCx, uint32_t Overrun)**
Function description	Set ADC group regular behavior in case of overrun: data preserved or overwritten.
Parameters	• **ADCx:** ADC instance • **Overrun:** This parameter can be one of the following values: – LL_ADC_REG_OVR_DATA_PRESERVED – LL_ADC_REG_OVR_DATA_OVERWRITTEN
Return values	• **None:**
Notes	• Compatibility with devices without feature overrun: other devices without this feature have a behavior equivalent to data overwritten. The default setting of overrun is data preserved. Therefore, for compatibility with all devices, parameter overrun should be set to data overwritten. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
Reference Manual to LL API cross reference:	• CFGR OVRMOD LL_ADC_REG_SetOverrun

LL_ADC_REG_GetOverrun

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_GetOverrun (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular behavior in case of overrun: data preserved or overwritten.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_REG_OVR_DATA_PRESERVED – LL_ADC_REG_OVR_DATA_OVERWRITTEN
Reference Manual to LL API cross reference:	• CFGR OVRMOD LL_ADC_REG_GetOverrun

LL_ADC_INJ_SetTriggerSource

Function name	**__STATIC_INLINE void LL_ADC_INJ_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)**
Function description	Set ADC group injected conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).
Parameters	• **ADCx:** ADC instance • **TriggerSource:** This parameter can be one of the following values: – LL_ADC_INJ_TRIG_SOFTWARE – LL_ADC_INJ_TRIG_EXT_TIM1_TRGO – LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2 – LL_ADC_INJ_TRIG_EXT_TIM1_CH4 – LL_ADC_INJ_TRIG_EXT_TIM2_TRGO – LL_ADC_INJ_TRIG_EXT_TIM2_CH1 – LL_ADC_INJ_TRIG_EXT_TIM3_TRGO – LL_ADC_INJ_TRIG_EXT_TIM3_CH1 – LL_ADC_INJ_TRIG_EXT_TIM3_CH3 – LL_ADC_INJ_TRIG_EXT_TIM3_CH4 – LL_ADC_INJ_TRIG_EXT_TIM4_TRGO – LL_ADC_INJ_TRIG_EXT_TIM6_TRGO – LL_ADC_INJ_TRIG_EXT_TIM8_CH4 – LL_ADC_INJ_TRIG_EXT_TIM8_TRGO – LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2 – LL_ADC_INJ_TRIG_EXT_TIM15_TRGO – LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
Return values	• **None:**
Notes	• On this STM32 serie, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_INJ_SetTriggerEdge(). • Availability of parameters of trigger sources from timer depends on timers availability on the selected device. • On this STM32 serie, setting of this feature is conditioned to

	ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• JSQR JEXTSEL LL_ADC_INJ_SetTriggerSource • JSQR JEXTEN LL_ADC_INJ_SetTriggerSource

LL_ADC_INJ_GetTriggerSource

Function name	**__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerSource (ADC_TypeDef * ADCx)**
Function description	Get ADC group injected conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_INJ_TRIG_SOFTWARE – LL_ADC_INJ_TRIG_EXT_TIM1_TRGO – LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2 – LL_ADC_INJ_TRIG_EXT_TIM1_CH4 – LL_ADC_INJ_TRIG_EXT_TIM2_TRGO – LL_ADC_INJ_TRIG_EXT_TIM2_CH1 – LL_ADC_INJ_TRIG_EXT_TIM3_TRGO – LL_ADC_INJ_TRIG_EXT_TIM3_CH1 – LL_ADC_INJ_TRIG_EXT_TIM3_CH3 – LL_ADC_INJ_TRIG_EXT_TIM3_CH4 – LL_ADC_INJ_TRIG_EXT_TIM4_TRGO – LL_ADC_INJ_TRIG_EXT_TIM6_TRGO – LL_ADC_INJ_TRIG_EXT_TIM8_CH4 – LL_ADC_INJ_TRIG_EXT_TIM8_TRGO – LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2 – LL_ADC_INJ_TRIG_EXT_TIM15_TRGO – LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
Notes	• To determine whether group injected trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_INJ_GetTriggerSource(ADC1) == LL_ADC_INJ_TRIG_SOFTWARE)") use function LL_ADC_INJ_IsTriggerSourceSWStart. • Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
Reference Manual to LL API cross reference:	• JSQR JEXTSEL LL_ADC_INJ_GetTriggerSource • JSQR JEXTEN LL_ADC_INJ_GetTriggerSource

LL_ADC_INJ_IsTriggerSourceSWStart

Function name	**__STATIC_INLINE uint32_t LL_ADC_INJ_IsTriggerSourceSWStart (ADC_TypeDef * ADCx)**
Function description	Get ADC group injected conversion trigger source internal (SW start) or external.

Parameters	• **ADCx:** ADC instance
Return values	• **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.
Notes	• In case of group injected trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_INJ_GetTriggerSource.
Reference Manual to LL API cross reference:	• JSQR JEXTEN LL_ADC_INJ_IsTriggerSourceSWStart

LL_ADC_INJ_SetTriggerEdge

Function name	**__STATIC_INLINE void LL_ADC_INJ_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)**
Function description	Set ADC group injected conversion trigger polarity.
Parameters	• **ADCx:** ADC instance • **ExternalTriggerEdge:** This parameter can be one of the following values: – LL_ADC_INJ_TRIG_EXT_RISING – LL_ADC_INJ_TRIG_EXT_FALLING – LL_ADC_INJ_TRIG_EXT_RISINGFALLING
Return values	• **None:**
Notes	• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• JSQR JEXTEN LL_ADC_INJ_SetTriggerEdge

LL_ADC_INJ_GetTriggerEdge

Function name	**__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerEdge (ADC_TypeDef * ADCx)**
Function description	Get ADC group injected conversion trigger polarity.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_INJ_TRIG_EXT_RISING – LL_ADC_INJ_TRIG_EXT_FALLING – LL_ADC_INJ_TRIG_EXT_RISINGFALLING
Reference Manual to LL API cross reference:	• JSQR JEXTEN LL_ADC_INJ_GetTriggerEdge

LL_ADC_INJ_SetSequencerLength

Function name	**__STATIC_INLINE void LL_ADC_INJ_SetSequencerLength**

	(ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)
Function description	Set ADC group injected sequencer length and scan direction.
Parameters	• **ADCx:** ADC instance • **SequencerNbRanks:** This parameter can be one of the following values: – LL_ADC_INJ_SEQ_SCAN_DISABLE – LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS – LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS – LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS
Return values	• **None:**
Notes	• This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). • Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• JSQR JL LL_ADC_INJ_SetSequencerLength

LL_ADC_INJ_GetSequencerLength

Function name	**__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerLength (ADC_TypeDef * ADCx)**
Function description	Get ADC group injected sequencer length and scan direction.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_INJ_SEQ_SCAN_DISABLE – LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS – LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS – LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS
Notes	• This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). • Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
Reference Manual to LL API cross reference:	• JSQR JL LL_ADC_INJ_GetSequencerLength

LL_ADC_INJ_SetSequencerDiscont

Function name	**__STATIC_INLINE void LL_ADC_INJ_SetSequencerDiscont**

	(ADC_TypeDef * ADCx, uint32_t SeqDiscont)
Function description	Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.
Parameters	• **ADCx:** ADC instance • **SeqDiscont:** This parameter can be one of the following values: – LL_ADC_INJ_SEQ_DISCONT_DISABLE – LL_ADC_INJ_SEQ_DISCONT_1RANK
Return values	• **None:**
Notes	• It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.
Reference Manual to LL API cross reference:	• CFGR JDISCEN LL_ADC_INJ_SetSequencerDiscont

LL_ADC_INJ_GetSequencerDiscont

Function name	**__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerDiscont (ADC_TypeDef * ADCx)**
Function description	Get ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_INJ_SEQ_DISCONT_DISABLE – LL_ADC_INJ_SEQ_DISCONT_1RANK
Reference Manual to LL API cross reference:	• CFGR JDISCEN LL_ADC_INJ_GetSequencerDiscont

LL_ADC_INJ_SetSequencerRanks

Function name	**__STATIC_INLINE void LL_ADC_INJ_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)**
Function description	Set ADC group injected sequence: channel on the selected sequence rank.
Parameters	• **ADCx:** ADC instance • **Rank:** This parameter can be one of the following values: – LL_ADC_INJ_RANK_1 – LL_ADC_INJ_RANK_2 – LL_ADC_INJ_RANK_3 – LL_ADC_INJ_RANK_4 • **Channel:** This parameter can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1. – LL_ADC_CHANNEL_0 – LL_ADC_CHANNEL_1 (7)

- LL_ADC_CHANNEL_2 (7)
- LL_ADC_CHANNEL_3 (7)
- LL_ADC_CHANNEL_4 (7)
- LL_ADC_CHANNEL_5 (7)
- LL_ADC_CHANNEL_6
- LL_ADC_CHANNEL_7
- LL_ADC_CHANNEL_8
- LL_ADC_CHANNEL_9
- LL_ADC_CHANNEL_10
- LL_ADC_CHANNEL_11
- LL_ADC_CHANNEL_12
- LL_ADC_CHANNEL_13
- LL_ADC_CHANNEL_14
- LL_ADC_CHANNEL_15
- LL_ADC_CHANNEL_16
- LL_ADC_CHANNEL_17
- LL_ADC_CHANNEL_18
- LL_ADC_CHANNEL_VREFINT (1)
- LL_ADC_CHANNEL_TEMPSENSOR (4)
- LL_ADC_CHANNEL_VBAT (4)
- LL_ADC_CHANNEL_DAC1CH1 (5)
- LL_ADC_CHANNEL_DAC1CH2 (5)
- LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
- LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
- LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
- LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

- (2) On STM32L4, parameter available only on ADC instance: ADC2.
- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3.
- (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.
- (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion rate up to 4.21 Ms/s)).

Return values

- **None:**

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On STM32L4, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN1..5).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with

		or without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• JSQR JSQ1 LL_ADC_INJ_SetSequencerRanks • JSQR JSQ2 LL_ADC_INJ_SetSequencerRanks • JSQR JSQ3 LL_ADC_INJ_SetSequencerRanks • JSQR JSQ4 LL_ADC_INJ_SetSequencerRanks	

LL_ADC_INJ_GetSequencerRanks

Function name	**__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank)**
Function description	Get ADC group injected sequence: channel on the selected sequence rank.
Parameters	• **ADCx:** ADC instance • **Rank:** This parameter can be one of the following values: – LL_ADC_INJ_RANK_1 – LL_ADC_INJ_RANK_2 – LL_ADC_INJ_RANK_3 – LL_ADC_INJ_RANK_4
Return values	• **Returned:** value can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1. – LL_ADC_CHANNEL_0 – LL_ADC_CHANNEL_1 (7) – LL_ADC_CHANNEL_2 (7) – LL_ADC_CHANNEL_3 (7) – LL_ADC_CHANNEL_4 (7) – LL_ADC_CHANNEL_5 (7) – LL_ADC_CHANNEL_6 – LL_ADC_CHANNEL_7 – LL_ADC_CHANNEL_8 – LL_ADC_CHANNEL_9 – LL_ADC_CHANNEL_10 – LL_ADC_CHANNEL_11 – LL_ADC_CHANNEL_12 – LL_ADC_CHANNEL_13 – LL_ADC_CHANNEL_14 – LL_ADC_CHANNEL_15 – LL_ADC_CHANNEL_16 – LL_ADC_CHANNEL_17 – LL_ADC_CHANNEL_18 – LL_ADC_CHANNEL_VREFINT (1) – LL_ADC_CHANNEL_TEMPSENSOR (4) – LL_ADC_CHANNEL_VBAT (4) – LL_ADC_CHANNEL_DAC1CH1 (5) – LL_ADC_CHANNEL_DAC1CH2 (5) – LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6) – LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6) – LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6) – LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6) • (2) On STM32L4, parameter available only on ADC instance: ADC2.

- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3.
- (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.
- (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion rate up to 4.21 Ms/s)).
- (1, 2, 3, 4) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_GetSequencerRanks

LL_ADC_INJ_SetTrigAuto

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTrigAuto (ADC_TypeDef * ADCx, uint32_t TrigAuto)

Function description

Set ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance
- **TrigAuto:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Return values

- **None:**

Notes

- This mode can be used to extend number of data registers updated after one ADC conversion trigger and with data permanently kept (not erased by successive conversions of scan of ADC sequencer ranks), up to 5 data registers: 1 data register on ADC group regular, 4 data registers on ADC group injected.

	• If ADC group injected injected trigger source is set to an external trigger, this feature must be must be set to independent trigger. ADC group injected automatic trigger is compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular. • It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• CFGR JAUTO LL_ADC_INJ_SetTrigAuto

LL_ADC_INJ_GetTrigAuto

Function name	**__STATIC_INLINE uint32_t LL_ADC_INJ_GetTrigAuto (ADC_TypeDef * ADCx)**
Function description	Get ADC group injected conversion trigger: independent or from ADC group regular.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_INJ_TRIG_INDEPENDENT – LL_ADC_INJ_TRIG_FROM_GRP_REGULAR
Reference Manual to LL API cross reference:	• CFGR JAUTO LL_ADC_INJ_GetTrigAuto

LL_ADC_INJ_SetQueueMode

Function name	**__STATIC_INLINE void LL_ADC_INJ_SetQueueMode (ADC_TypeDef * ADCx, uint32_t QueueMode)**
Function description	Set ADC group injected contexts queue mode.
Parameters	• **ADCx:** ADC instance • **QueueMode:** This parameter can be one of the following values: – LL_ADC_INJ_QUEUE_DISABLE – LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE – LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY
Return values	• **None:**
Notes	• A context is a setting of group injected sequencer: group injected triggersequencer lengthsequencer ranks If contexts queue is disabled:only 1 sequence can be configured and is active perpetually. If contexts queue is enabled:up to 2 contexts can be queued and are checked in and out as a FIFO stack (first-in, first-out).If a new context is set when queues is full, error is triggered by interruption "Injected Queue Overflow".Two behaviors are possible when all contexts have

been processed: the contexts queue can maintain the last context active perpetually or can be empty and injected group triggers are disabled.Triggers can be only external (not internal SW start)Caution: The sequence must be fully configured in one time (one write of register JSQR makes a check-in of a new context into the queue). Therefore functions to set separately injected trigger and sequencer channels cannot be used, register JSQR must be set using function LL_ADC_INJ_ConfigQueueContext().

- This parameter can be modified only when no conversion is on going on either groups regular or injected.
- A modification of the context mode (bit JQDIS) causes the contexts queue to be flushed and the register JSQR is cleared.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:
- CFGR JQM LL_ADC_INJ_SetQueueMode
- CFGR JQDIS LL_ADC_INJ_SetQueueMode

LL_ADC_INJ_GetQueueMode

Function name
__STATIC_INLINE uint32_t LL_ADC_INJ_GetQueueMode (ADC_TypeDef * ADCx)

Function description
Get ADC group injected context queue mode.

Parameters
- **ADCx:** ADC instance

Return values
- **Returned:** value can be one of the following values:
  – LL_ADC_INJ_QUEUE_DISABLE
  – LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE
  – LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

Reference Manual to LL API cross reference:
- CFGR JQM LL_ADC_INJ_GetQueueMode
- CFGR JQDIS LL_ADC_INJ_GetQueueMode

LL_ADC_INJ_ConfigQueueContext

Function name
__STATIC_INLINE void LL_ADC_INJ_ConfigQueueContext (ADC_TypeDef * ADCx, uint32_t TriggerSource, uint32_t ExternalTriggerEdge, uint32_t SequencerNbRanks, uint32_t Rank1_Channel, uint32_t Rank2_Channel, uint32_t Rank3_Channel, uint32_t Rank4_Channel)

Function description
Set one context on ADC group injected that will be checked in contexts queue.

Parameters
- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  – LL_ADC_INJ_TRIG_SOFTWARE
  – LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  – LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  – LL_ADC_INJ_TRIG_EXT_TIM2_TRGO

  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
- **ExternalTriggerEdge:** This parameter can be one of the following values: Note: This parameter is discarded in case of SW start: parameter "TriggerSource" set to "LL_ADC_INJ_TRIG_SOFTWARE".
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS
- **Rank1_Channel:** This parameter can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (7)
  - LL_ADC_CHANNEL_2 (7)
  - LL_ADC_CHANNEL_3 (7)
  - LL_ADC_CHANNEL_4 (7)
  - LL_ADC_CHANNEL_5 (7)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (4)
  - LL_ADC_CHANNEL_VBAT (4)
  - LL_ADC_CHANNEL_DAC1CH1 (5)
  - LL_ADC_CHANNEL_DAC1CH2 (5)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)

  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)
- (2) On STM32L4, parameter available only on ADC instance: ADC2.
- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3.
- (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.
- (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion rate up to 4.21 Ms/s)).
- **Rank2_Channel:** This parameter can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (7)
  - LL_ADC_CHANNEL_2 (7)
  - LL_ADC_CHANNEL_3 (7)
  - LL_ADC_CHANNEL_4 (7)
  - LL_ADC_CHANNEL_5 (7)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (4)
  - LL_ADC_CHANNEL_VBAT (4)
  - LL_ADC_CHANNEL_DAC1CH1 (5)
  - LL_ADC_CHANNEL_DAC1CH2 (5)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)
- (2) On STM32L4, parameter available only on ADC instance: ADC2.
- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC

instances: ADC1, ADC3.

- (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.
- (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion rate up to 4.21 Ms/s)).
- **Rank3_Channel:** This parameter can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (7)
  - LL_ADC_CHANNEL_2 (7)
  - LL_ADC_CHANNEL_3 (7)
  - LL_ADC_CHANNEL_4 (7)
  - LL_ADC_CHANNEL_5 (7)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (4)
  - LL_ADC_CHANNEL_VBAT (4)
  - LL_ADC_CHANNEL_DAC1CH1 (5)
  - LL_ADC_CHANNEL_DAC1CH2 (5)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)
- (2) On STM32L4, parameter available only on ADC instance: ADC2.
- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3.
- (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.
- (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion

rate up to 4.21 Ms/s)).

- **Rank4_Channel:** This parameter can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1.
  – LL_ADC_CHANNEL_0
  – LL_ADC_CHANNEL_1 (7)
  – LL_ADC_CHANNEL_2 (7)
  – LL_ADC_CHANNEL_3 (7)
  – LL_ADC_CHANNEL_4 (7)
  – LL_ADC_CHANNEL_5 (7)
  – LL_ADC_CHANNEL_6
  – LL_ADC_CHANNEL_7
  – LL_ADC_CHANNEL_8
  – LL_ADC_CHANNEL_9
  – LL_ADC_CHANNEL_10
  – LL_ADC_CHANNEL_11
  – LL_ADC_CHANNEL_12
  – LL_ADC_CHANNEL_13
  – LL_ADC_CHANNEL_14
  – LL_ADC_CHANNEL_15
  – LL_ADC_CHANNEL_16
  – LL_ADC_CHANNEL_17
  – LL_ADC_CHANNEL_18
  – LL_ADC_CHANNEL_VREFINT (1)
  – LL_ADC_CHANNEL_TEMPSENSOR (4)
  – LL_ADC_CHANNEL_VBAT (4)
  – LL_ADC_CHANNEL_DAC1CH1 (5)
  – LL_ADC_CHANNEL_DAC1CH2 (5)
  – LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  – LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  – LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  – LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)
- (2) On STM32L4, parameter available only on ADC instance: ADC2.
- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3.
- (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.
- (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion rate up to 4.21 Ms/s)).

Return values

- **None:**

Notes

- A context is a setting of group injected sequencer: group injected triggersequencer lengthsequencer ranks This function is intended to be used when contexts queue is enabled, because the sequence must be fully configured in one time (functions to set separately injected trigger and sequencer channels cannot be used): Refer to function LL_ADC_INJ_SetQueueMode().

- In the contexts queue, only the active context can be read. The parameters of this function can be read using functions: LL_ADC_INJ_GetTriggerSource() LL_ADC_INJ_GetTriggerEdge() LL_ADC_INJ_GetSequencerRanks()
- On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On STM32L4, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN1..5).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_ConfigQueueContext
- JSQR JEXTEN LL_ADC_INJ_ConfigQueueContext
- JSQR JL LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ1 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ2 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ3 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ4 LL_ADC_INJ_ConfigQueueContext

LL_ADC_SetChannelSamplingTime

Function name

__STATIC_INLINE void LL_ADC_SetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SamplingTime)

Function description

Set sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1.
  – LL_ADC_CHANNEL_0
  – LL_ADC_CHANNEL_1 (7)
  – LL_ADC_CHANNEL_2 (7)
  – LL_ADC_CHANNEL_3 (7)
  – LL_ADC_CHANNEL_4 (7)
  – LL_ADC_CHANNEL_5 (7)
  – LL_ADC_CHANNEL_6
  – LL_ADC_CHANNEL_7
  – LL_ADC_CHANNEL_8
  – LL_ADC_CHANNEL_9
  – LL_ADC_CHANNEL_10
  – LL_ADC_CHANNEL_11
  – LL_ADC_CHANNEL_12
  – LL_ADC_CHANNEL_13
  – LL_ADC_CHANNEL_14
  – LL_ADC_CHANNEL_15

  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (4)
  - LL_ADC_CHANNEL_VBAT (4)
  - LL_ADC_CHANNEL_DAC1CH1 (5)
  - LL_ADC_CHANNEL_DAC1CH2 (5)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)
- (2) On STM32L4, parameter available only on ADC instance: ADC2.
- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3.
- (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.
- (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion rate up to 4.21 Ms/s)).
- **SamplingTime:** This parameter can be one of the following values: (1) On some devices, ADC sampling time 2.5 ADC clock cycles can be replaced by 3.5 ADC clock cycles. Refer to function LL_ADC_SetSamplingTimeCommonConfig().
  - LL_ADC_SAMPLINGTIME_2CYCLES_5 (1)
  - LL_ADC_SAMPLINGTIME_6CYCLES_5
  - LL_ADC_SAMPLINGTIME_12CYCLES_5
  - LL_ADC_SAMPLINGTIME_24CYCLES_5
  - LL_ADC_SAMPLINGTIME_47CYCLES_5
  - LL_ADC_SAMPLINGTIME_92CYCLES_5
  - LL_ADC_SAMPLINGTIME_247CYCLES_5
  - LL_ADC_SAMPLINGTIME_640CYCLES_5

Return values

- **None:**

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- In case of internal channel (VrefInt, TempSensor, ...) to be converted: sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values (parameters TS_vrefint, TS_temp, ...).
- Conversion time is the addition of sampling time and processing time. On this STM32 serie, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution

6 bits

- In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_SetChannelSamplingTime

LL_ADC_GetChannelSamplingTime

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values: (1) On STM32L4, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (7)
  - LL_ADC_CHANNEL_2 (7)
  - LL_ADC_CHANNEL_3 (7)
  - LL_ADC_CHANNEL_4 (7)
  - LL_ADC_CHANNEL_5 (7)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12

- – LL_ADC_CHANNEL_13
- – LL_ADC_CHANNEL_14
- – LL_ADC_CHANNEL_15
- – LL_ADC_CHANNEL_16
- – LL_ADC_CHANNEL_17
- – LL_ADC_CHANNEL_18
- – LL_ADC_CHANNEL_VREFINT (1)
- – LL_ADC_CHANNEL_TEMPSENSOR (4)
- – LL_ADC_CHANNEL_VBAT (4)
- – LL_ADC_CHANNEL_DAC1CH1 (5)
- – LL_ADC_CHANNEL_DAC1CH2 (5)
- – LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
- – LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
- – LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
- – LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

- (2) On STM32L4, parameter available only on ADC instance: ADC2.
- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3.
- (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.
- (7) On STM32L4, fast channel (0.188 us for 12-bit resolution (ADC conversion rate up to 5.33 Ms/s)). Other channels are slow channels (0.238 us for 12-bit resolution (ADC conversion rate up to 4.21 Ms/s)).

Return values

- **Returned:** value can be one of the following values: (1) On some devices, ADC sampling time 2.5 ADC clock cycles can be replaced by 3.5 ADC clock cycles. Refer to function LL_ADC_SetSamplingTimeCommonConfig().
  - – LL_ADC_SAMPLINGTIME_2CYCLES_5 (1)
  - – LL_ADC_SAMPLINGTIME_6CYCLES_5
  - – LL_ADC_SAMPLINGTIME_12CYCLES_5
  - – LL_ADC_SAMPLINGTIME_24CYCLES_5
  - – LL_ADC_SAMPLINGTIME_47CYCLES_5
  - – LL_ADC_SAMPLINGTIME_92CYCLES_5
  - – LL_ADC_SAMPLINGTIME_247CYCLES_5
  - – LL_ADC_SAMPLINGTIME_640CYCLES_5

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- Conversion time is the addition of sampling time and processing time. On this STM32 serie, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits

Reference Manual to LL API cross

- SMPR1 SMP0 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_GetChannelSamplingTime

reference:	• SMPR1 SMP2 LL_ADC_GetChannelSamplingTime • SMPR1 SMP3 LL_ADC_GetChannelSamplingTime • SMPR1 SMP4 LL_ADC_GetChannelSamplingTime • SMPR1 SMP5 LL_ADC_GetChannelSamplingTime • SMPR1 SMP6 LL_ADC_GetChannelSamplingTime • SMPR1 SMP7 LL_ADC_GetChannelSamplingTime • SMPR1 SMP8 LL_ADC_GetChannelSamplingTime • SMPR1 SMP9 LL_ADC_GetChannelSamplingTime • SMPR2 SMP10 LL_ADC_GetChannelSamplingTime • SMPR2 SMP11 LL_ADC_GetChannelSamplingTime • SMPR2 SMP12 LL_ADC_GetChannelSamplingTime • SMPR2 SMP13 LL_ADC_GetChannelSamplingTime • SMPR2 SMP14 LL_ADC_GetChannelSamplingTime • SMPR2 SMP15 LL_ADC_GetChannelSamplingTime • SMPR2 SMP16 LL_ADC_GetChannelSamplingTime • SMPR2 SMP17 LL_ADC_GetChannelSamplingTime • SMPR2 SMP18 LL_ADC_GetChannelSamplingTime

LL_ADC_SetChannelSingleDiff

Function name	**__STATIC_INLINE void LL_ADC_SetChannelSingleDiff (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SingleDiff)**
Function description	Set mode single-ended or differential input of the selected ADC channel.
Parameters	• **ADCx:** ADC instance • **Channel:** This parameter can be one of the following values: – LL_ADC_CHANNEL_1 – LL_ADC_CHANNEL_2 – LL_ADC_CHANNEL_3 – LL_ADC_CHANNEL_4 – LL_ADC_CHANNEL_5 – LL_ADC_CHANNEL_6 – LL_ADC_CHANNEL_7 – LL_ADC_CHANNEL_8 – LL_ADC_CHANNEL_9 – LL_ADC_CHANNEL_10 – LL_ADC_CHANNEL_11 – LL_ADC_CHANNEL_12 – LL_ADC_CHANNEL_13 – LL_ADC_CHANNEL_14 • **SingleDiff:** This parameter can be a combination of the following values: – LL_ADC_SINGLE_ENDED – LL_ADC_DIFFERENTIAL_ENDED
Return values	• **None:**
Notes	• Channel ending is on channel scope: independently of channel mapped on ADC group regular or injected. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically.

- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- On STM32L4, channels 15, 16, 17, 18 of ADC1, ADC2, ADC3 (if available) are internally fixed to single-ended inputs configuration.
- For ADC channels configured in differential mode, both inputs should be biased at (Vref+)/2 +/-200mV. (Vref+ is the analog voltage reference)
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:
- DIFSEL DIFSEL LL_ADC_GetChannelSamplingTime

LL_ADC_GetChannelSingleDiff

Function name
__STATIC_INLINE uint32_t LL_ADC_GetChannelSingleDiff (ADC_TypeDef * ADCx, uint32_t Channel)

Function description
Get mode single-ended or differential input of the selected ADC channel.

Parameters
- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
  – LL_ADC_CHANNEL_1
  – LL_ADC_CHANNEL_2
  – LL_ADC_CHANNEL_3
  – LL_ADC_CHANNEL_4
  – LL_ADC_CHANNEL_5
  – LL_ADC_CHANNEL_6
  – LL_ADC_CHANNEL_7
  – LL_ADC_CHANNEL_8
  – LL_ADC_CHANNEL_9
  – LL_ADC_CHANNEL_10
  – LL_ADC_CHANNEL_11
  – LL_ADC_CHANNEL_12
  – LL_ADC_CHANNEL_13
  – LL_ADC_CHANNEL_14

Return values
- **0:** channel in single-ended mode, else: channel in differential mode

Notes
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Therefore, to ensure a channel is configured in single-ended mode, the configuration of channel itself and the channel 'i-1' must be read back (to ensure that the selected channel channel has not been configured in differential mode by the previous channel).

- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- On STM32L4, channels 15, 16, 17, 18 of ADC1, ADC2, ADC3 (if available) are internally fixed to single-ended inputs configuration.
- One or several values can be selected. In this case, the value returned is null if all channels are in single ended-mode. Example: (LL_ADC_CHANNEL_4 **|** LL_ADC_CHANNEL_12 **|** ...)

Reference Manual to LL API cross reference:
- DIFSEL DIFSEL LL_ADC_GetChannelSamplingTime

LL_ADC_SetAnalogWDMonitChannels

Function name
: **__STATIC_INLINE void LL_ADC_SetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDChannelGroup)**

Function description
: Set ADC analog watchdog monitored channels: a single channel, multiple channels or all channels, on ADC groups regular and-or injected.

Parameters
- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  – LL_ADC_AWD1
  – LL_ADC_AWD2
  – LL_ADC_AWD3
- **AWDChannelGroup:** This parameter can be one of the following values: (0) On STM32L4, parameter available only on analog watchdog number: AWD1.
  – LL_ADC_AWD_DISABLE
  – LL_ADC_AWD_ALL_CHANNELS_REG (0)
  – LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  – LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  – LL_ADC_AWD_CHANNEL_0_REG (0)
  – LL_ADC_AWD_CHANNEL_0_INJ (0)
  – LL_ADC_AWD_CHANNEL_0_REG_INJ
  – LL_ADC_AWD_CHANNEL_1_REG (0)
  – LL_ADC_AWD_CHANNEL_1_INJ (0)
  – LL_ADC_AWD_CHANNEL_1_REG_INJ
  – LL_ADC_AWD_CHANNEL_2_REG (0)
  – LL_ADC_AWD_CHANNEL_2_INJ (0)
  – LL_ADC_AWD_CHANNEL_2_REG_INJ
  – LL_ADC_AWD_CHANNEL_3_REG (0)
  – LL_ADC_AWD_CHANNEL_3_INJ (0)
  – LL_ADC_AWD_CHANNEL_3_REG_INJ
  – LL_ADC_AWD_CHANNEL_4_REG (0)
  – LL_ADC_AWD_CHANNEL_4_INJ (0)
  – LL_ADC_AWD_CHANNEL_4_REG_INJ

- LL_ADC_AWD_CHANNEL_5_REG (0)
- LL_ADC_AWD_CHANNEL_5_INJ (0)
- LL_ADC_AWD_CHANNEL_5_REG_INJ
- LL_ADC_AWD_CHANNEL_6_REG (0)
- LL_ADC_AWD_CHANNEL_6_INJ (0)
- LL_ADC_AWD_CHANNEL_6_REG_INJ
- LL_ADC_AWD_CHANNEL_7_REG (0)
- LL_ADC_AWD_CHANNEL_7_INJ (0)
- LL_ADC_AWD_CHANNEL_7_REG_INJ
- LL_ADC_AWD_CHANNEL_8_REG (0)
- LL_ADC_AWD_CHANNEL_8_INJ (0)
- LL_ADC_AWD_CHANNEL_8_REG_INJ
- LL_ADC_AWD_CHANNEL_9_REG (0)
- LL_ADC_AWD_CHANNEL_9_INJ (0)
- LL_ADC_AWD_CHANNEL_9_REG_INJ
- LL_ADC_AWD_CHANNEL_10_REG (0)
- LL_ADC_AWD_CHANNEL_10_INJ (0)
- LL_ADC_AWD_CHANNEL_10_REG_INJ
- LL_ADC_AWD_CHANNEL_11_REG (0)
- LL_ADC_AWD_CHANNEL_11_INJ (0)
- LL_ADC_AWD_CHANNEL_11_REG_INJ
- LL_ADC_AWD_CHANNEL_12_REG (0)
- LL_ADC_AWD_CHANNEL_12_INJ (0)
- LL_ADC_AWD_CHANNEL_12_REG_INJ
- LL_ADC_AWD_CHANNEL_13_REG (0)
- LL_ADC_AWD_CHANNEL_13_INJ (0)
- LL_ADC_AWD_CHANNEL_13_REG_INJ
- LL_ADC_AWD_CHANNEL_14_REG (0)
- LL_ADC_AWD_CHANNEL_14_INJ (0)
- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG (0)
- LL_ADC_AWD_CHANNEL_15_INJ (0)
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG (0)
- LL_ADC_AWD_CHANNEL_16_INJ (0)
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG (0)
- LL_ADC_AWD_CHANNEL_17_INJ (0)
- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CHANNEL_18_REG (0)
- LL_ADC_AWD_CHANNEL_18_INJ (0)
- LL_ADC_AWD_CHANNEL_18_REG_INJ
- LL_ADC_AWD_CH_VREFINT_REG (0)(1)
- LL_ADC_AWD_CH_VREFINT_INJ (0)(1)
- LL_ADC_AWD_CH_VREFINT_REG_INJ (1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG (0)(4)
- LL_ADC_AWD_CH_TEMPSENSOR_INJ (0)(4)
- LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ (4)
- LL_ADC_AWD_CH_VBAT_REG (0)(4)
- LL_ADC_AWD_CH_VBAT_INJ (0)(4)
- LL_ADC_AWD_CH_VBAT_REG_INJ (4)
- LL_ADC_AWD_CH_DAC1CH1_REG (0)(2)(5)

- – LL_ADC_AWD_CH_DAC1CH1_INJ (0)(2)(5)
- – LL_ADC_AWD_CH_DAC1CH1_REG_INJ (2)(5)
- – LL_ADC_AWD_CH_DAC1CH2_REG (0)(2)(5)
- – LL_ADC_AWD_CH_DAC1CH2_INJ (0)(2)(5)
- – LL_ADC_AWD_CH_DAC1CH2_REG_INJ (2)(5)
- – LL_ADC_AWD_CH_DAC1CH1_ADC2_REG (0)(2)(6)
- – LL_ADC_AWD_CH_DAC1CH1_ADC2_INJ (0)(2)(6)
- – LL_ADC_AWD_CH_DAC1CH1_ADC2_REG_INJ (2)(6)
- – LL_ADC_AWD_CH_DAC1CH2_ADC2_REG (0)(2)(6)
- – LL_ADC_AWD_CH_DAC1CH2_ADC2_INJ (0)(2)(6)
- – LL_ADC_AWD_CH_DAC1CH2_ADC2_REG_INJ (2)(6)
- – LL_ADC_AWD_CH_DAC1CH1_ADC3_REG (0)(3)(6)
- – LL_ADC_AWD_CH_DAC1CH1_ADC3_INJ (0)(3)(6)
- – LL_ADC_AWD_CH_DAC1CH1_ADC3_REG_INJ (3)(6)
- – LL_ADC_AWD_CH_DAC1CH2_ADC3_REG (0)(3)(6)
- – LL_ADC_AWD_CH_DAC1CH2_ADC3_INJ (0)(3)(6)
- – LL_ADC_AWD_CH_DAC1CH2_ADC3_REG_INJ (3)(6)

- (1) On STM32L4, parameter available only on ADC instance: ADC1.
- (2) On STM32L4, parameter available only on ADC instance: ADC2.
- (3) On STM32L4, parameter available only on ADC instance: ADC3.
- (4) On STM32L4, parameter available only on ADC instances: ADC1, ADC3. (5) On STM32L4, parameter available on devices with only 1 ADC instance.
- (6) On STM32L4, parameter available on devices with several ADC instances.

Return values

- **None:**

Notes

- Once monitored channels are selected, analog watchdog is enabled.
- In case of need to define a single channel to monitor with analog watchdog from sequencer channel definition, use helper macro __LL_ADC_ANALOGWD_CHANNEL_GROUP().
- On this STM32 serie, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.

	• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• CFGR AWD1CH LL_ADC_SetAnalogWDMonitChannels • CFGR AWD1SGL LL_ADC_SetAnalogWDMonitChannels • CFGR AWD1EN LL_ADC_SetAnalogWDMonitChannels • CFGR JAWD1EN LL_ADC_SetAnalogWDMonitChannels • AWD2CR AWD2CH LL_ADC_SetAnalogWDMonitChannels • AWD3CR AWD3CH LL_ADC_SetAnalogWDMonitChannels

LL_ADC_GetAnalogWDMonitChannels

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDy)**
Function description	Get ADC analog watchdog monitored channel.
Parameters	• **ADCx:** ADC instance • **AWDy:** This parameter can be one of the following values: (1) On this AWD number, monitored channel can be retrieved if only 1 channel is programmed (or none or all channels). This function cannot retrieve monitored channel if multiple channels are programmed simultaneously by bitfield. – LL_ADC_AWD1 – LL_ADC_AWD2 (1) – LL_ADC_AWD3 (1)
Return values	• **Returned:** value can be one of the following values: (0) On STM32L4, parameter available only on analog watchdog number: AWD1. – LL_ADC_AWD_DISABLE – LL_ADC_AWD_ALL_CHANNELS_REG (0) – LL_ADC_AWD_ALL_CHANNELS_INJ (0) – LL_ADC_AWD_ALL_CHANNELS_REG_INJ – LL_ADC_AWD_CHANNEL_0_REG (0) – LL_ADC_AWD_CHANNEL_0_INJ (0) – LL_ADC_AWD_CHANNEL_0_REG_INJ – LL_ADC_AWD_CHANNEL_1_REG (0) – LL_ADC_AWD_CHANNEL_1_INJ (0) – LL_ADC_AWD_CHANNEL_1_REG_INJ – LL_ADC_AWD_CHANNEL_2_REG (0) – LL_ADC_AWD_CHANNEL_2_INJ (0) – LL_ADC_AWD_CHANNEL_2_REG_INJ – LL_ADC_AWD_CHANNEL_3_REG (0) – LL_ADC_AWD_CHANNEL_3_INJ (0) – LL_ADC_AWD_CHANNEL_3_REG_INJ – LL_ADC_AWD_CHANNEL_4_REG (0) – LL_ADC_AWD_CHANNEL_4_INJ (0) – LL_ADC_AWD_CHANNEL_4_REG_INJ – LL_ADC_AWD_CHANNEL_5_REG (0) – LL_ADC_AWD_CHANNEL_5_INJ (0) – LL_ADC_AWD_CHANNEL_5_REG_INJ – LL_ADC_AWD_CHANNEL_6_REG (0)

- LL_ADC_AWD_CHANNEL_6_INJ (0)
- LL_ADC_AWD_CHANNEL_6_REG_INJ
- LL_ADC_AWD_CHANNEL_7_REG (0)
- LL_ADC_AWD_CHANNEL_7_INJ (0)
- LL_ADC_AWD_CHANNEL_7_REG_INJ
- LL_ADC_AWD_CHANNEL_8_REG (0)
- LL_ADC_AWD_CHANNEL_8_INJ (0)
- LL_ADC_AWD_CHANNEL_8_REG_INJ
- LL_ADC_AWD_CHANNEL_9_REG (0)
- LL_ADC_AWD_CHANNEL_9_INJ (0)
- LL_ADC_AWD_CHANNEL_9_REG_INJ
- LL_ADC_AWD_CHANNEL_10_REG (0)
- LL_ADC_AWD_CHANNEL_10_INJ (0)
- LL_ADC_AWD_CHANNEL_10_REG_INJ
- LL_ADC_AWD_CHANNEL_11_REG (0)
- LL_ADC_AWD_CHANNEL_11_INJ (0)
- LL_ADC_AWD_CHANNEL_11_REG_INJ
- LL_ADC_AWD_CHANNEL_12_REG (0)
- LL_ADC_AWD_CHANNEL_12_INJ (0)
- LL_ADC_AWD_CHANNEL_12_REG_INJ
- LL_ADC_AWD_CHANNEL_13_REG (0)
- LL_ADC_AWD_CHANNEL_13_INJ (0)
- LL_ADC_AWD_CHANNEL_13_REG_INJ
- LL_ADC_AWD_CHANNEL_14_REG (0)
- LL_ADC_AWD_CHANNEL_14_INJ (0)
- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG (0)
- LL_ADC_AWD_CHANNEL_15_INJ (0)
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG (0)
- LL_ADC_AWD_CHANNEL_16_INJ (0)
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG (0)
- LL_ADC_AWD_CHANNEL_17_INJ (0)
- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CHANNEL_18_REG (0)
- LL_ADC_AWD_CHANNEL_18_INJ (0)
- LL_ADC_AWD_CHANNEL_18_REG_INJ

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Applicable only when the analog watchdog is set to monitor one channel.
- On this STM32 serie, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups

	monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• CFGR AWD1CH LL_ADC_GetAnalogWDMonitChannels • CFGR AWD1SGL LL_ADC_GetAnalogWDMonitChannels • CFGR AWD1EN LL_ADC_GetAnalogWDMonitChannels • CFGR JAWD1EN LL_ADC_GetAnalogWDMonitChannels • AWD2CR AWD2CH LL_ADC_GetAnalogWDMonitChannels • AWD3CR AWD3CH LL_ADC_GetAnalogWDMonitChannels

LL_ADC_ConfigAnalogWDThresholds

Function name	**__STATIC_INLINE void LL_ADC_ConfigAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdHighValue, uint32_t AWDThresholdLowValue)**
Function description	Set ADC analog watchdog thresholds value of both thresholds high and low.
Parameters	• **ADCx:** ADC instance • **AWDy:** This parameter can be one of the following values: – LL_ADC_AWD1 – LL_ADC_AWD2 – LL_ADC_AWD3 • **AWDThresholdHighValue:** Value between Min_Data=0x000 and Max_Data=0xFFF • **AWDThresholdLowValue:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None:**
Notes	• If value of only one threshold high or low must be set, use function LL_ADC_SetAnalogWDThresholds(). • In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(). • On this STM32 serie, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups

monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_ConfigAnalogWDThresholds
- TR2 HT2 LL_ADC_ConfigAnalogWDThresholds
- TR3 HT3 LL_ADC_ConfigAnalogWDThresholds
- TR1 LT1 LL_ADC_ConfigAnalogWDThresholds
- TR2 LT2 LL_ADC_ConfigAnalogWDThresholds
- TR3 LT3 LL_ADC_ConfigAnalogWDThresholds

LL_ADC_SetAnalogWDThresholds

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow, uint32_t AWDThresholdValue)

Function description

Set ADC analog watchdog threshold value of threshold high or low.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
- **AWDThresholdValue:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- If values of both thresholds high or low must be set, use function LL_ADC_ConfigAnalogWDThresholds().
- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTIO N().
- On this STM32 serie, there are 2 kinds of analog watchdog

	instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either ADC groups regular or injected.
Reference Manual to LL API cross reference:	• TR1 HT1 LL_ADC_SetAnalogWDThresholds • TR2 HT2 LL_ADC_SetAnalogWDThresholds • TR3 HT3 LL_ADC_SetAnalogWDThresholds • TR1 LT1 LL_ADC_SetAnalogWDThresholds • TR2 LT2 LL_ADC_SetAnalogWDThresholds • TR3 LT3 LL_ADC_SetAnalogWDThresholds

LL_ADC_GetAnalogWDThresholds

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow)**
Function description	Get ADC analog watchdog threshold value of threshold high, threshold low or raw data with ADC thresholds high and low concatenated.
Parameters	• **ADCx:** ADC instance • **AWDy:** This parameter can be one of the following values: – LL_ADC_AWD1 – LL_ADC_AWD2 – LL_ADC_AWD3 • **AWDThresholdsHighLow:** This parameter can be one of the following values: – LL_ADC_AWD_THRESHOLD_HIGH – LL_ADC_AWD_THRESHOLD_LOW – LL_ADC_AWD_THRESHOLDS_HIGH_LOW
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0xFFF
Notes	• If raw data with ADC thresholds high and low is retrieved, the data of each threshold high or low can be isolated using helper macro: __LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW(). • In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper

	macro __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION().
Reference Manual to LL API cross reference:	• TR1 HT1 LL_ADC_GetAnalogWDThresholds • TR2 HT2 LL_ADC_GetAnalogWDThresholds • TR3 HT3 LL_ADC_GetAnalogWDThresholds • TR1 LT1 LL_ADC_GetAnalogWDThresholds • TR2 LT2 LL_ADC_GetAnalogWDThresholds • TR3 LT3 LL_ADC_GetAnalogWDThresholds

LL_ADC_SetOverSamplingScope

Function name	**__STATIC_INLINE void LL_ADC_SetOverSamplingScope (ADC_TypeDef * ADCx, uint32_t OvsScope)**
Function description	Set ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 families).
Parameters	• **ADCx:** ADC instance • **OvsScope:** This parameter can be one of the following values: – LL_ADC_OVS_DISABLE – LL_ADC_OVS_GRP_REGULAR_CONTINUED – LL_ADC_OVS_GRP_REGULAR_RESUMED – LL_ADC_OVS_GRP_INJECTED – LL_ADC_OVS_GRP_INJ_REG_RESUMED
Return values	• **None:**
Notes	• If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset). • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• CFGR2 ROVSE LL_ADC_SetOverSamplingScope • CFGR2 JOVSE LL_ADC_SetOverSamplingScope • CFGR2 ROVSM LL_ADC_SetOverSamplingScope

LL_ADC_GetOverSamplingScope

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingScope (ADC_TypeDef * ADCx)**
Function description	Get ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 families).
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_OVS_DISABLE – LL_ADC_OVS_GRP_REGULAR_CONTINUED – LL_ADC_OVS_GRP_REGULAR_RESUMED – LL_ADC_OVS_GRP_INJECTED

– LL_ADC_OVS_GRP_INJ_REG_RESUMED

Notes	• If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).
Reference Manual to LL API cross reference:	• CFGR2 ROVSE LL_ADC_GetOverSamplingScope • CFGR2 JOVSE LL_ADC_GetOverSamplingScope • CFGR2 ROVSM LL_ADC_GetOverSamplingScope

LL_ADC_SetOverSamplingDiscont

Function name	**__STATIC_INLINE void LL_ADC_SetOverSamplingDiscont (ADC_TypeDef * ADCx, uint32_t OverSamplingDiscont)**
Function description	Set ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.
Parameters	• **ADCx:** ADC instance • **OverSamplingDiscont:** This parameter can be one of the following values: – LL_ADC_OVS_REG_CONT – LL_ADC_OVS_REG_DISCONT
Return values	• **None:**
Notes	• Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger) • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular. • On this STM32 serie, oversampling discontinuous mode (triggered mode) can be used only when oversampling is set on group regular only and in resumed mode.
Reference Manual to LL API cross reference:	• CFGR2 TROVS LL_ADC_SetOverSamplingDiscont

LL_ADC_GetOverSamplingDiscont

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingDiscont (ADC_TypeDef * ADCx)**
Function description	Get ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_OVS_REG_CONT – LL_ADC_OVS_REG_DISCONT
Notes	• Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are

	done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)
Reference Manual to LL API cross reference:	• CFGR2 TROVS LL_ADC_GetOverSamplingDiscont

LL_ADC_ConfigOverSamplingRatioShift

Function name	**__STATIC_INLINE void LL_ADC_ConfigOverSamplingRatioShift (ADC_TypeDef * ADCx, uint32_t Ratio, uint32_t Shift)**
Function description	Set ADC oversampling (impacting both ADC groups regular and injected)
Parameters	• **ADCx:** ADC instance • **Ratio:** This parameter can be one of the following values: – LL_ADC_OVS_RATIO_2 – LL_ADC_OVS_RATIO_4 – LL_ADC_OVS_RATIO_8 – LL_ADC_OVS_RATIO_16 – LL_ADC_OVS_RATIO_32 – LL_ADC_OVS_RATIO_64 – LL_ADC_OVS_RATIO_128 – LL_ADC_OVS_RATIO_256 • **Shift:** This parameter can be one of the following values: – LL_ADC_OVS_SHIFT_NONE – LL_ADC_OVS_SHIFT_RIGHT_1 – LL_ADC_OVS_SHIFT_RIGHT_2 – LL_ADC_OVS_SHIFT_RIGHT_3 – LL_ADC_OVS_SHIFT_RIGHT_4 – LL_ADC_OVS_SHIFT_RIGHT_5 – LL_ADC_OVS_SHIFT_RIGHT_6 – LL_ADC_OVS_SHIFT_RIGHT_7 – LL_ADC_OVS_SHIFT_RIGHT_8
Return values	• **None:**
Notes	• This function set the 2 items of oversampling configuration: ratioshift • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• CFGR2 OVSS LL_ADC_ConfigOverSamplingRatioShift • CFGR2 OVSR LL_ADC_ConfigOverSamplingRatioShift

LL_ADC_GetOverSamplingRatio

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingRatio (ADC_TypeDef * ADCx)**
Function description	Get ADC oversampling ratio (impacting both ADC groups regular and injected)

Parameters	• **ADCx:** ADC instance
Return values	• **Ratio:** This parameter can be one of the following values: – LL_ADC_OVS_RATIO_2 – LL_ADC_OVS_RATIO_4 – LL_ADC_OVS_RATIO_8 – LL_ADC_OVS_RATIO_16 – LL_ADC_OVS_RATIO_32 – LL_ADC_OVS_RATIO_64 – LL_ADC_OVS_RATIO_128 – LL_ADC_OVS_RATIO_256
Reference Manual to LL API cross reference:	• CFGR2 OVSR LL_ADC_GetOverSamplingRatio

LL_ADC_GetOverSamplingShift

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingShift (ADC_TypeDef * ADCx)**
Function description	Get ADC oversampling shift (impacting both ADC groups regular and injected)
Parameters	• **ADCx:** ADC instance
Return values	• **Shift:** This parameter can be one of the following values: – LL_ADC_OVS_SHIFT_NONE – LL_ADC_OVS_SHIFT_RIGHT_1 – LL_ADC_OVS_SHIFT_RIGHT_2 – LL_ADC_OVS_SHIFT_RIGHT_3 – LL_ADC_OVS_SHIFT_RIGHT_4 – LL_ADC_OVS_SHIFT_RIGHT_5 – LL_ADC_OVS_SHIFT_RIGHT_6 – LL_ADC_OVS_SHIFT_RIGHT_7 – LL_ADC_OVS_SHIFT_RIGHT_8
Reference Manual to LL API cross reference:	• CFGR2 OVSS LL_ADC_GetOverSamplingShift

LL_ADC_REG_SetTrigSource

Function name	**__STATIC_INLINE void LL_ADC_REG_SetTrigSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)**
Function description	

LL_ADC_INJ_SetTrigSource

Function name	**__STATIC_INLINE void LL_ADC_INJ_SetTrigSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)**
Function description	

LL_ADC_EnableDeepPowerDown

Function name	**__STATIC_INLINE void LL_ADC_EnableDeepPowerDown (ADC_TypeDef * ADCx)**
Function description	Put ADC instance in deep power down state.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
Reference Manual to LL API cross reference:	• CR DEEPPWD LL_ADC_EnableDeepPowerDown

LL_ADC_DisableDeepPowerDown

Function name	**__STATIC_INLINE void LL_ADC_DisableDeepPowerDown (ADC_TypeDef * ADCx)**
Function description	Disable ADC deep power down mode.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
Reference Manual to LL API cross reference:	• CR DEEPPWD LL_ADC_DisableDeepPowerDown

LL_ADC_IsDeepPowerDownEnabled

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsDeepPowerDownEnabled (ADC_TypeDef * ADCx)**
Function description	Get the selected ADC instance deep power down state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** deep power down is disabled, 1: deep power down is enabled.
Reference Manual to LL API cross reference:	• CR DEEPPWD LL_ADC_IsDeepPowerDownEnabled

LL_ADC_EnableInternalRegulator

Function name	**__STATIC_INLINE void LL_ADC_EnableInternalRegulator (ADC_TypeDef * ADCx)**
Function description	Enable ADC instance internal voltage regulator.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, after ADC internal voltage regulator enable, a delay for ADC internal voltage regulator stabilization is required before performing a ADC calibration or ADC enable. Refer to device datasheet, parameter tADCVREG_STUP. Refer to literal LL_ADC_DELAY_INTERNAL_REGUL_STAB_US. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
Reference Manual to LL API cross reference:	• CR ADVREGEN LL_ADC_EnableInternalRegulator

LL_ADC_DisableInternalRegulator

Function name	**__STATIC_INLINE void LL_ADC_DisableInternalRegulator (ADC_TypeDef * ADCx)**
Function description	Disable ADC internal voltage regulator.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
Reference Manual to LL API cross reference:	• CR ADVREGEN LL_ADC_DisableInternalRegulator

LL_ADC_IsInternalRegulatorEnabled

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsInternalRegulatorEnabled (ADC_TypeDef * ADCx)**
Function description	Get the selected ADC instance internal voltage regulator state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** internal regulator is disabled, 1: internal regulator is enabled.
Reference Manual to LL API cross reference:	• CR ADVREGEN LL_ADC_IsInternalRegulatorEnabled

LL_ADC_Enable

Function name	**__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef ***

	ADCx)
Function description	Enable the selected ADC instance.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, after ADC enable, a delay for ADC internal analog stabilization is required before performing a ADC conversion start. Refer to device datasheet, parameter tSTAB. • On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain) • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled and ADC internal voltage regulator enabled.
Reference Manual to LL API cross reference:	• CR ADEN LL_ADC_Enable

LL_ADC_Disable

Function name	**__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)**
Function description	Disable the selected ADC instance.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be not disabled. Must be enabled without conversion on going on either groups regular or injected.
Reference Manual to LL API cross reference:	• CR ADDIS LL_ADC_Disable

LL_ADC_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabled (ADC_TypeDef * ADCx)**
Function description	Get the selected ADC instance enable state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** ADC is disabled, 1: ADC is enabled.
Notes	• On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)
Reference Manual to LL API cross	• CR ADEN LL_ADC_IsEnabled

reference:

LL_ADC_IsDisableOngoing

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsDisableOngoing (ADC_TypeDef * ADCx)**
Function description	Get the selected ADC instance disable state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** no ADC disable command on going.
Reference Manual to LL API cross reference:	• CR ADDIS LL_ADC_IsDisableOngoing

LL_ADC_StartCalibration

Function name	**__STATIC_INLINE void LL_ADC_StartCalibration (ADC_TypeDef * ADCx, uint32_t SingleDiff)**
Function description	Start ADC calibration in the mode single-ended or differential (for devices with differential mode available).
Parameters	• **ADCx:** ADC instance • **SingleDiff:** This parameter can be one of the following values: – LL_ADC_SINGLE_ENDED – LL_ADC_DIFFERENTIAL_ENDED
Return values	• **None:**
Notes	• On this STM32 serie, a minimum number of ADC clock cycles are required between ADC end of calibration and ADC enable. Refer to literal LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES. • For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration run must be performed for each of these differential modes, if used afterwards and if the application requires their calibration). • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
Reference Manual to LL API cross reference:	• CR ADCAL LL_ADC_StartCalibration • CR ADCALDIF LL_ADC_StartCalibration

LL_ADC_IsCalibrationOnGoing

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsCalibrationOnGoing (ADC_TypeDef * ADCx)**
Function description	Get ADC calibration state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** calibration complete, 1: calibration in progress.

Reference Manual to LL API cross reference:	• CR ADCAL LL_ADC_IsCalibrationOnGoing

LL_ADC_REG_StartConversion

Function name	**__STATIC_INLINE void LL_ADC_REG_StartConversion (ADC_TypeDef * ADCx)**
Function description	Start ADC group regular conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.
Reference Manual to LL API cross reference:	• CR ADSTART LL_ADC_REG_StartConversion

LL_ADC_REG_StopConversion

Function name	**__STATIC_INLINE void LL_ADC_REG_StopConversion (ADC_TypeDef * ADCx)**
Function description	Stop ADC group regular conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group regular, without ADC disable command on going.
Reference Manual to LL API cross reference:	• CR ADSTP LL_ADC_REG_StopConversion

LL_ADC_REG_IsConversionOngoing

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_IsConversionOngoing (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion state.
Parameters	• **ADCx:** ADC instance

Return values	• **0:** no conversion is on going on ADC group regular.
Reference Manual to LL API cross reference:	• CR ADSTART LL_ADC_REG_IsConversionOngoing

LL_ADC_REG_IsStopConversionOngoing

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_IsStopConversionOngoing (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular command of conversion stop state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** no command of conversion stop is on going on ADC group regular.
Reference Manual to LL API cross reference:	• CR ADSTP LL_ADC_REG_IsStopConversionOngoing

LL_ADC_REG_ReadConversionData32

Function name	**__STATIC_INLINE uint32_t LL_ADC_REG_ReadConversionData32 (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).
Parameters	• **ADCx:** ADC instance
Return values	• **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF
Reference Manual to LL API cross reference:	• DR RDATA LL_ADC_REG_ReadConversionData32

LL_ADC_REG_ReadConversionData12

Function name	**__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData12 (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data, range fit for ADC resolution 12 bits.
Parameters	• **ADCx:** ADC instance
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0xFFF
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross	• DR RDATA LL_ADC_REG_ReadConversionData12

reference:

LL_ADC_REG_ReadConversionData10

Function name	**__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData10 (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data, range fit for ADC resolution 10 bits.
Parameters	• **ADCx:** ADC instance
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0x3FF
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross reference:	• DR RDATA LL_ADC_REG_ReadConversionData10

LL_ADC_REG_ReadConversionData8

Function name	**__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData8 (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data, range fit for ADC resolution 8 bits.
Parameters	• **ADCx:** ADC instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross reference:	• DR RDATA LL_ADC_REG_ReadConversionData8

LL_ADC_REG_ReadConversionData6

Function name	**__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData6 (ADC_TypeDef * ADCx)**
Function description	Get ADC group regular conversion data, range fit for ADC resolution 6 bits.
Parameters	• **ADCx:** ADC instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x3F
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.
Reference Manual to LL API cross	• DR RDATA LL_ADC_REG_ReadConversionData6

reference:

LL_ADC_INJ_StartConversion

Function name	**__STATIC_INLINE void LL_ADC_INJ_StartConversion (ADC_TypeDef * ADCx)**
Function description	Start ADC group injected conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command. • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group injected, without conversion stop command on going on group injected, without ADC disable command on going.
Reference Manual to LL API cross reference:	• CR JADSTART LL_ADC_INJ_StartConversion

LL_ADC_INJ_StopConversion

Function name	**__STATIC_INLINE void LL_ADC_INJ_StopConversion (ADC_TypeDef * ADCx)**
Function description	Stop ADC group injected conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group injected, without ADC disable command on going.
Reference Manual to LL API cross reference:	• CR JADSTP LL_ADC_INJ_StopConversion

LL_ADC_INJ_IsConversionOngoing

Function name	**__STATIC_INLINE uint32_t LL_ADC_INJ_IsConversionOngoing (ADC_TypeDef * ADCx)**
Function description	Get ADC group injected conversion state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** no conversion is on going on ADC group injected.
Reference Manual to	• CR JADSTART LL_ADC_INJ_IsConversionOngoing

LL API cross reference:

LL_ADC_INJ_IsStopConversionOngoing

Function name	**__STATIC_INLINE uint32_t LL_ADC_INJ_IsStopConversionOngoing (ADC_TypeDef * ADCx)**
Function description	Get ADC group injected command of conversion stop state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** no command of conversion stop is on going on ADC group injected.
Reference Manual to LL API cross reference:	• CR JADSTP LL_ADC_INJ_IsStopConversionOngoing

LL_ADC_INJ_ReadConversionData32

Function name	**__STATIC_INLINE uint32_t LL_ADC_INJ_ReadConversionData32 (ADC_TypeDef * ADCx, uint32_t Rank)**
Function description	Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).
Parameters	• **ADCx:** ADC instance • **Rank:** This parameter can be one of the following values: – LL_ADC_INJ_RANK_1 – LL_ADC_INJ_RANK_2 – LL_ADC_INJ_RANK_3 – LL_ADC_INJ_RANK_4
Return values	• **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF
Reference Manual to LL API cross reference:	• JDR1 JDATA LL_ADC_INJ_ReadConversionData32 • JDR2 JDATA LL_ADC_INJ_ReadConversionData32 • JDR3 JDATA LL_ADC_INJ_ReadConversionData32 • JDR4 JDATA LL_ADC_INJ_ReadConversionData32

LL_ADC_INJ_ReadConversionData12

Function name	**__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData12 (ADC_TypeDef * ADCx, uint32_t Rank)**
Function description	Get ADC group injected conversion data, range fit for ADC resolution 12 bits.
Parameters	• **ADCx:** ADC instance • **Rank:** This parameter can be one of the following values: – LL_ADC_INJ_RANK_1 – LL_ADC_INJ_RANK_2 – LL_ADC_INJ_RANK_3

	– LL_ADC_INJ_RANK_4
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0xFFF
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.
Reference Manual to LL API cross reference:	• JDR1 JDATA LL_ADC_INJ_ReadConversionData12 • JDR2 JDATA LL_ADC_INJ_ReadConversionData12 • JDR3 JDATA LL_ADC_INJ_ReadConversionData12 • JDR4 JDATA LL_ADC_INJ_ReadConversionData12

LL_ADC_INJ_ReadConversionData10

Function name	**__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData10 (ADC_TypeDef * ADCx, uint32_t Rank)**
Function description	Get ADC group injected conversion data, range fit for ADC resolution 10 bits.
Parameters	• **ADCx:** ADC instance • **Rank:** This parameter can be one of the following values: – LL_ADC_INJ_RANK_1 – LL_ADC_INJ_RANK_2 – LL_ADC_INJ_RANK_3 – LL_ADC_INJ_RANK_4
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0x3FF
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.
Reference Manual to LL API cross reference:	• JDR1 JDATA LL_ADC_INJ_ReadConversionData10 • JDR2 JDATA LL_ADC_INJ_ReadConversionData10 • JDR3 JDATA LL_ADC_INJ_ReadConversionData10 • JDR4 JDATA LL_ADC_INJ_ReadConversionData10

LL_ADC_INJ_ReadConversionData8

Function name	**__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData8 (ADC_TypeDef * ADCx, uint32_t Rank)**
Function description	Get ADC group injected conversion data, range fit for ADC resolution 8 bits.
Parameters	• **ADCx:** ADC instance • **Rank:** This parameter can be one of the following values: – LL_ADC_INJ_RANK_1 – LL_ADC_INJ_RANK_2 – LL_ADC_INJ_RANK_3 – LL_ADC_INJ_RANK_4
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Notes	• For devices with feature oversampling: Oversampling can

increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:	• JDR1 JDATA LL_ADC_INJ_ReadConversionData8 • JDR2 JDATA LL_ADC_INJ_ReadConversionData8 • JDR3 JDATA LL_ADC_INJ_ReadConversionData8 • JDR4 JDATA LL_ADC_INJ_ReadConversionData8

LL_ADC_INJ_ReadConversionData6

Function name	**__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData6 (ADC_TypeDef * ADCx, uint32_t Rank)**
Function description	Get ADC group injected conversion data, range fit for ADC resolution 6 bits.
Parameters	• **ADCx:** ADC instance • **Rank:** This parameter can be one of the following values: – LL_ADC_INJ_RANK_1 – LL_ADC_INJ_RANK_2 – LL_ADC_INJ_RANK_3 – LL_ADC_INJ_RANK_4
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x3F
Notes	• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.
Reference Manual to LL API cross reference:	• JDR1 JDATA LL_ADC_INJ_ReadConversionData6 • JDR2 JDATA LL_ADC_INJ_ReadConversionData6 • JDR3 JDATA LL_ADC_INJ_ReadConversionData6 • JDR4 JDATA LL_ADC_INJ_ReadConversionData6

LL_ADC_IsActiveFlag_ADRDY

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_ADRDY (ADC_TypeDef * ADCx)**
Function description	Get flag ADC ready.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Notes	• On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)
Reference Manual to LL API cross reference:	• ISR ADRDY LL_ADC_IsActiveFlag_ADRDY

LL_ADC_IsActiveFlag_EOC

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOC (ADC_TypeDef * ADCx)**

Function description	Get flag ADC group regular end of unitary conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR EOC LL_ADC_IsActiveFlag_EOC

LL_ADC_IsActiveFlag_EOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOS (ADC_TypeDef * ADCx)**
Function description	Get flag ADC group regular end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR EOS LL_ADC_IsActiveFlag_EOS

LL_ADC_IsActiveFlag_OVR

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_OVR (ADC_TypeDef * ADCx)**
Function description	Get flag ADC group regular overrun.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR OVR LL_ADC_IsActiveFlag_OVR

LL_ADC_IsActiveFlag_EOSMP

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOSMP (ADC_TypeDef * ADCx)**
Function description	Get flag ADC group regular end of sampling phase.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR EOSMP LL_ADC_IsActiveFlag_EOSMP

LL_ADC_IsActiveFlag_JEOC

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOC (ADC_TypeDef * ADCx)**
Function description	Get flag ADC group injected end of unitary conversion.

Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR JEOC LL_ADC_IsActiveFlag_JEOC

LL_ADC_IsActiveFlag_JEOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOS (ADC_TypeDef * ADCx)**
Function description	Get flag ADC group injected end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR JEOS LL_ADC_IsActiveFlag_JEOS

LL_ADC_IsActiveFlag_JQOVF

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JQOVF (ADC_TypeDef * ADCx)**
Function description	Get flag ADC group injected contexts queue overflow.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR JQOVF LL_ADC_IsActiveFlag_JQOVF

LL_ADC_IsActiveFlag_AWD1

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD1 (ADC_TypeDef * ADCx)**
Function description	Get flag ADC analog watchdog 1 flag.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR AWD1 LL_ADC_IsActiveFlag_AWD1

LL_ADC_IsActiveFlag_AWD2

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD2 (ADC_TypeDef * ADCx)**
Function description	Get flag ADC analog watchdog 2.
Parameters	• **ADCx:** ADC instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR AWD2 LL_ADC_IsActiveFlag_AWD2

LL_ADC_IsActiveFlag_AWD3

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD3 (ADC_TypeDef * ADCx)**
Function description	Get flag ADC analog watchdog 3.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR AWD3 LL_ADC_IsActiveFlag_AWD3

LL_ADC_ClearFlag_ADRDY

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_ADRDY (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC ready.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)
Reference Manual to LL API cross reference:	• ISR ADRDY LL_ADC_ClearFlag_ADRDY

LL_ADC_ClearFlag_EOC

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_EOC (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group regular end of unitary conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR EOC LL_ADC_ClearFlag_EOC

LL_ADC_ClearFlag_EOS

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_EOS (ADC_TypeDef * ADCx)**

Function description	Clear flag ADC group regular end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR EOS LL_ADC_ClearFlag_EOS

LL_ADC_ClearFlag_OVR

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_OVR (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group regular overrun.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR OVR LL_ADC_ClearFlag_OVR

LL_ADC_ClearFlag_EOSMP

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_EOSMP (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group regular end of sampling phase.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR EOSMP LL_ADC_ClearFlag_EOSMP

LL_ADC_ClearFlag_JEOC

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_JEOC (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group injected end of unitary conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR JEOC LL_ADC_ClearFlag_JEOC

LL_ADC_ClearFlag_JEOS

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_JEOS (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group injected end of sequence conversions.

Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR JEOS LL_ADC_ClearFlag_JEOS

LL_ADC_ClearFlag_JQOVF

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_JQOVF (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group injected contexts queue overflow.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR JQOVF LL_ADC_ClearFlag_JQOVF

LL_ADC_ClearFlag_AWD1

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_AWD1 (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC analog watchdog 1.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR AWD1 LL_ADC_ClearFlag_AWD1

LL_ADC_ClearFlag_AWD2

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_AWD2 (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC analog watchdog 2.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR AWD2 LL_ADC_ClearFlag_AWD2

LL_ADC_ClearFlag_AWD3

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_AWD3 (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC analog watchdog 3.
Parameters	• **ADCx:** ADC instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR AWD3 LL_ADC_ClearFlag_AWD3

LL_ADC_EnableIT_ADRDY

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_ADRDY (ADC_TypeDef * ADCx)**
Function description	Enable ADC ready.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER ADRDYIE LL_ADC_EnableIT_ADRDY

LL_ADC_EnableIT_EOC

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_EOC (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group regular end of unitary conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER EOCIE LL_ADC_EnableIT_EOC

LL_ADC_EnableIT_EOS

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group regular end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER EOSIE LL_ADC_EnableIT_EOS

LL_ADC_EnableIT_OVR

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_OVR (ADC_TypeDef * ADCx)**
Function description	Enable ADC group regular interruption overrun.
Parameters	• **ADCx:** ADC instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• IER OVRIE LL_ADC_EnableIT_OVR

LL_ADC_EnableIT_EOSMP

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_EOSMP (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group regular end of sampling.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER EOSMPIE LL_ADC_EnableIT_EOSMP

LL_ADC_EnableIT_JEOC

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_JEOC (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group injected end of unitary conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER JEOCIE LL_ADC_EnableIT_JEOC

LL_ADC_EnableIT_JEOS

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_JEOS (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group injected end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER JEOSIE LL_ADC_EnableIT_JEOS

LL_ADC_EnableIT_JQOVF

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_JQOVF (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group injected context queue overflow.
Parameters	• **ADCx:** ADC instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• IER JQOVFIE LL_ADC_EnableIT_JQOVF

LL_ADC_EnableIT_AWD1

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_AWD1 (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC analog watchdog 1.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER AWD1IE LL_ADC_EnableIT_AWD1

LL_ADC_EnableIT_AWD2

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_AWD2 (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC analog watchdog 2.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER AWD2IE LL_ADC_EnableIT_AWD2

LL_ADC_EnableIT_AWD3

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_AWD3 (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC analog watchdog 3.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER AWD3IE LL_ADC_EnableIT_AWD3

LL_ADC_DisableIT_ADRDY

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_ADRDY (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC ready.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• IER ADRDYIE LL_ADC_DisableIT_ADRDY

LL_ADC_DisableIT_EOC

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_EOC (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group regular end of unitary conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER EOCIE LL_ADC_DisableIT_EOC

LL_ADC_DisableIT_EOS

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_EOS (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group regular end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER EOSIE LL_ADC_DisableIT_EOS

LL_ADC_DisableIT_OVR

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_OVR (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group regular overrun.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER OVRIE LL_ADC_DisableIT_OVR

LL_ADC_DisableIT_EOSMP

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_EOSMP (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group regular end of sampling.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• IER EOSMPIE LL_ADC_DisableIT_EOSMP

LL_ADC_DisableIT_JEOC

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_JEOC (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group regular end of unitary conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER JEOCIE LL_ADC_DisableIT_JEOC

LL_ADC_DisableIT_JEOS

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_JEOS (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group injected end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER JEOSIE LL_ADC_DisableIT_JEOS

LL_ADC_DisableIT_JQOVF

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_JQOVF (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group injected context queue overflow.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER JQOVFIE LL_ADC_DisableIT_JQOVF

LL_ADC_DisableIT_AWD1

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_AWD1 (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC analog watchdog 1.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• IER AWD1IE LL_ADC_DisableIT_AWD1

LL_ADC_DisableIT_AWD2

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_AWD2 (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC analog watchdog 2.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER AWD2IE LL_ADC_DisableIT_AWD2

LL_ADC_DisableIT_AWD3

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_AWD3 (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC analog watchdog 3.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER AWD3IE LL_ADC_DisableIT_AWD3

LL_ADC_IsEnabledIT_ADRDY

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_ADRDY (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC ready (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER ADRDYIE LL_ADC_IsEnabledIT_ADRDY

LL_ADC_IsEnabledIT_EOC

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOC (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group regular end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• IER EOCIE LL_ADC_IsEnabledIT_EOC

LL_ADC_IsEnabledIT_EOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group regular end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER EOSIE LL_ADC_IsEnabledIT_EOS

LL_ADC_IsEnabledIT_OVR

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_OVR (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group regular overrun (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER OVRIE LL_ADC_IsEnabledIT_OVR

LL_ADC_IsEnabledIT_EOSMP

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOSMP (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group regular end of sampling (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER EOSMPIE LL_ADC_IsEnabledIT_EOSMP

LL_ADC_IsEnabledIT_JEOC

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOC (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group injected end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER JEOCIE LL_ADC_IsEnabledIT_JEOC

LL_ADC_IsEnabledIT_JEOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOS (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group injected end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER JEOSIE LL_ADC_IsEnabledIT_JEOS

LL_ADC_IsEnabledIT_JQOVF

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JQOVF (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group injected context queue overflow interrupt state (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER JQOVFIE LL_ADC_IsEnabledIT_JQOVF

LL_ADC_IsEnabledIT_AWD1

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD1 (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC analog watchdog 1 (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER AWD1IE LL_ADC_IsEnabledIT_AWD1

LL_ADC_IsEnabledIT_AWD2

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD2 (ADC_TypeDef * ADCx)**
Function description	Get state of interruption Get ADC analog watchdog 2 (0: interrupt disabled, 1: interrupt enabled).

Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER AWD2IE LL_ADC_IsEnabledIT_AWD2

LL_ADC_IsEnabledIT_AWD3

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD3 (ADC_TypeDef * ADCx)**
Function description	Get state of interruption Get ADC analog watchdog 3 (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER AWD3IE LL_ADC_IsEnabledIT_AWD3

LL_ADC_CommonDeInit

Function name	**ErrorStatus LL_ADC_CommonDeInit (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	De-initialize registers of all ADC instances belonging to the same ADC common instance to their default reset values.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC common registers are de-initialized – ERROR: not applicable
Notes	• This function is performing a hard reset, using high level clock source RCC ADC reset. Caution: On this STM32 serie, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. To de-initialize only 1 ADC instance, use function LL_ADC_DeInit().

LL_ADC_CommonInit

Function name	**ErrorStatus LL_ADC_CommonInit (ADC_Common_TypeDef * ADCxy_COMMON, LL_ADC_CommonInitTypeDef * ADC_CommonInitStruct)**
Function description	Initialize some features of ADC common parameters (all ADC instances belonging to the same ADC common instance) and multimode (for devices with several ADC instances available).
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

	• **ADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC common registers are initialized – ERROR: ADC common registers are not initialized
Notes	• The setting of ADC common parameters is conditioned to ADC instances state: All ADC instances belonging to the same ADC common instance must be disabled.

LL_ADC_CommonStructInit

Function name	**void LL_ADC_CommonStructInit (LL_ADC_CommonInitTypeDef * ADC_CommonInitStruct)**
Function description	Set each LL_ADC_CommonInitTypeDef field to default value.
Parameters	• **ADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

LL_ADC_DeInit

Function name	**ErrorStatus LL_ADC_DeInit (ADC_TypeDef * ADCx)**
Function description	De-initialize registers of the selected ADC instance to their default reset values.
Parameters	• **ADCx:** ADC instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC registers are de-initialized – ERROR: ADC registers are not de-initialized
Notes	• To reset all ADC instances quickly (perform a hard reset), use function LL_ADC_CommonDeInit(). • If this functions returns error status, it means that ADC instance is in an unknown state. In this case, perform a hard reset using high level clock source RCC ADC reset. Caution: On this STM32 serie, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. Refer to function LL_ADC_CommonDeInit().

LL_ADC_Init

Function name	**ErrorStatus LL_ADC_Init (ADC_TypeDef * ADCx, LL_ADC_InitTypeDef * ADC_InitStruct)**
Function description	Initialize some features of ADC instance.
Parameters	• **ADCx:** ADC instance • **ADC_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC registers are initialized

	– ERROR: ADC registers are not initialized
Notes	• These parameters have an impact on ADC scope: ADC instance. Affects both group regular and group injected (availability of ADC group injected depends on STM32 families). Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: ADC instance . • The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state. • After using this function, some other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_StructInit

Function name	**void LL_ADC_StructInit (LL_ADC_InitTypeDef * ADC_InitStruct)**
Function description	Set each LL_ADC_InitTypeDef field to default value.
Parameters	• **ADC_InitStruct:** Pointer to a LL_ADC_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

LL_ADC_REG_Init

Function name	**ErrorStatus LL_ADC_REG_Init (ADC_TypeDef * ADCx, LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)**
Function description	Initialize some features of ADC group regular.
Parameters	• **ADCx:** ADC instance • **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC registers are initialized – ERROR: ADC registers are not initialized
Notes	• These parameters have an impact on ADC scope: ADC group regular. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "REG").

- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_REG_StructInit

Function name	**void LL_ADC_REG_StructInit (LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)**
Function description	Set each LL_ADC_REG_InitTypeDef field to default value.
Parameters	• **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

LL_ADC_INJ_Init

Function name	**ErrorStatus LL_ADC_INJ_Init (ADC_TypeDef * ADCx, LL_ADC_INJ_InitTypeDef * ADC_INJ_InitStruct)**
Function description	Initialize some features of ADC group injected.
Parameters	• **ADCx:** ADC instance • **ADC_INJ_InitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC registers are initialized – ERROR: ADC registers are not initialized
Notes	• These parameters have an impact on ADC scope: ADC group injected. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "INJ"). • The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on

going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.

- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_INJ_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_INJ_StructInit

Function name	**void LL_ADC_INJ_StructInit (LL_ADC_INJ_InitTypeDef * ADC_INJ_InitStruct)**
Function description	Set each LL_ADC_INJ_InitTypeDef field to default value.
Parameters	• **ADC_INJ_InitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

73.3 ADC Firmware driver defines

73.3.1 ADC

Analog watchdog - Monitored channels

LL_ADC_AWD_DISABLE	ADC analog watchdog monitoring disabled
LL_ADC_AWD_ALL_CHANNELS_REG	ADC analog watchdog monitoring of all channels, converted by group regular only
LL_ADC_AWD_ALL_CHANNELS_INJ	ADC analog watchdog monitoring of all channels, converted by group injected only
LL_ADC_AWD_ALL_CHANNELS_REG_INJ	ADC analog watchdog monitoring of all channels, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_0_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group regular only
LL_ADC_AWD_CHANNEL_0_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group injected only
LL_ADC_AWD_CHANNEL_0_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0,

	converted by either group regular or injected
LL_ADC_AWD_CHANNEL_1_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group regular only
LL_ADC_AWD_CHANNEL_1_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group injected only
LL_ADC_AWD_CHANNEL_1_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_2_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by group regular only
LL_ADC_AWD_CHANNEL_2_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by group injected only
LL_ADC_AWD_CHANNEL_2_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_3_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group regular only
LL_ADC_AWD_CHANNEL_3_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group injected only
LL_ADC_AWD_CHANNEL_3_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_4_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group regular only
LL_ADC_AWD_CHANNEL_4_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group injected only

LL_ADC_AWD_CHANNEL_4_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_5_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group regular only
LL_ADC_AWD_CHANNEL_5_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group injected only
LL_ADC_AWD_CHANNEL_5_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_6_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group regular only
LL_ADC_AWD_CHANNEL_6_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group injected only
LL_ADC_AWD_CHANNEL_6_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_7_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by group regular only
LL_ADC_AWD_CHANNEL_7_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by group injected only
LL_ADC_AWD_CHANNEL_7_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_8_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by group regular only
LL_ADC_AWD_CHANNEL_8_INJ	ADC analog watchdog monitoring of ADC external channel (channel

	connected to GPIO pin) ADCx_IN8, converted by group injected only
LL_ADC_AWD_CHANNEL_8_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_9_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by group regular only
LL_ADC_AWD_CHANNEL_9_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by group injected only
LL_ADC_AWD_CHANNEL_9_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_10_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by group regular only
LL_ADC_AWD_CHANNEL_10_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by group injected only
LL_ADC_AWD_CHANNEL_10_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_11_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by group regular only
LL_ADC_AWD_CHANNEL_11_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by group injected only
LL_ADC_AWD_CHANNEL_11_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_12_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12,

	converted by group regular only
LL_ADC_AWD_CHANNEL_12_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by group injected only
LL_ADC_AWD_CHANNEL_12_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_13_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by group regular only
LL_ADC_AWD_CHANNEL_13_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by group injected only
LL_ADC_AWD_CHANNEL_13_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_14_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by group regular only
LL_ADC_AWD_CHANNEL_14_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by group injected only
LL_ADC_AWD_CHANNEL_14_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_15_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by group regular only
LL_ADC_AWD_CHANNEL_15_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by group injected only
LL_ADC_AWD_CHANNEL_15_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_16_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by group regular only
LL_ADC_AWD_CHANNEL_16_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by group injected only
LL_ADC_AWD_CHANNEL_16_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_17_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group regular only
LL_ADC_AWD_CHANNEL_17_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group injected only
LL_ADC_AWD_CHANNEL_17_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_18_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by group regular only
LL_ADC_AWD_CHANNEL_18_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by group injected only
LL_ADC_AWD_CHANNEL_18_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by either group regular or injected
LL_ADC_AWD_CH_VREFINT_REG	ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group regular only
LL_ADC_AWD_CH_VREFINT_INJ	ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group injected only
LL_ADC_AWD_CH_VREFINT_REG_INJ	ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference,

	converted by either group regular or injected
LL_ADC_AWD_CH_TEMPSENSOR_REG	ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group regular only
LL_ADC_AWD_CH_TEMPSENSOR_INJ	ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group injected only
LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ	ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by either group regular or injected
LL_ADC_AWD_CH_VBAT_REG	ADC analog watchdog monitoring of ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda, converted by group regular only
LL_ADC_AWD_CH_VBAT_INJ	ADC analog watchdog monitoring of ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda, converted by group injected only
LL_ADC_AWD_CH_VBAT_REG_INJ	ADC analog watchdog monitoring of ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda
LL_ADC_AWD_CH_DAC1CH1_REG	ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC1, converted by group regular only
LL_ADC_AWD_CH_DAC1CH1_INJ	ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC1, converted by group injected only
LL_ADC_AWD_CH_DAC1CH1_REG_INJ	ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC1, converted by either group regular or injected
LL_ADC_AWD_CH_DAC1CH2_REG	ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC1, converted by group regular only
LL_ADC_AWD_CH_DAC1CH2_INJ	ADC analog watchdog monitoring of ADC internal channel connected to

	DAC1 channel 1, channel specific to ADC1, converted by group injected only
LL_ADC_AWD_CH_DAC1CH2_REG_INJ	ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC1, converted by either group regular or injected

Analog watchdog - Analog watchdog number

LL_ADC_AWD1	ADC analog watchdog number 1
LL_ADC_AWD2	ADC analog watchdog number 2
LL_ADC_AWD3	ADC analog watchdog number 3

Analog watchdog - Thresholds

LL_ADC_AWD_THRESHOLD_HIGH	ADC analog watchdog threshold high
LL_ADC_AWD_THRESHOLD_LOW	ADC analog watchdog threshold low
LL_ADC_AWD_THRESHOLDS_HIGH_LOW	ADC analog watchdog both thresholds high and low concatenated into the same data

ADC instance - Channel number

LL_ADC_CHANNEL_0	ADC external channel (channel connected to GPIO pin) ADCx_IN0
LL_ADC_CHANNEL_1	ADC external channel (channel connected to GPIO pin) ADCx_IN1
LL_ADC_CHANNEL_2	ADC external channel (channel connected to GPIO pin) ADCx_IN2
LL_ADC_CHANNEL_3	ADC external channel (channel connected to GPIO pin) ADCx_IN3
LL_ADC_CHANNEL_4	ADC external channel (channel connected to GPIO pin) ADCx_IN4
LL_ADC_CHANNEL_5	ADC external channel (channel connected to GPIO pin) ADCx_IN5
LL_ADC_CHANNEL_6	ADC external channel (channel connected to GPIO pin) ADCx_IN6
LL_ADC_CHANNEL_7	ADC external channel (channel connected to GPIO pin) ADCx_IN7
LL_ADC_CHANNEL_8	ADC external channel (channel connected to GPIO pin) ADCx_IN8
LL_ADC_CHANNEL_9	ADC external channel (channel connected to GPIO pin) ADCx_IN9
LL_ADC_CHANNEL_10	ADC external channel (channel connected to GPIO pin) ADCx_IN10
LL_ADC_CHANNEL_11	ADC external channel (channel connected to GPIO pin) ADCx_IN11
LL_ADC_CHANNEL_12	ADC external channel (channel connected to GPIO pin) ADCx_IN12

LL_ADC_CHANNEL_13	ADC external channel (channel connected to GPIO pin) ADCx_IN13
LL_ADC_CHANNEL_14	ADC external channel (channel connected to GPIO pin) ADCx_IN14
LL_ADC_CHANNEL_15	ADC external channel (channel connected to GPIO pin) ADCx_IN15
LL_ADC_CHANNEL_16	ADC external channel (channel connected to GPIO pin) ADCx_IN16
LL_ADC_CHANNEL_17	ADC external channel (channel connected to GPIO pin) ADCx_IN17
LL_ADC_CHANNEL_18	ADC external channel (channel connected to GPIO pin) ADCx_IN18
LL_ADC_CHANNEL_VREFINT	ADC internal channel connected to VrefInt: Internal voltage reference. On STM32L4, ADC channel available only on ADC instance: ADC1.
LL_ADC_CHANNEL_TEMPSENSOR	ADC internal channel connected to Temperature sensor. On STM32L4, ADC channel available only on ADC instances: ADC1, ADC3.
LL_ADC_CHANNEL_VBAT	ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda. On STM32L4, ADC channel available only on ADC instances: ADC1, ADC3.
LL_ADC_CHANNEL_DAC1CH1	ADC internal channel connected to DAC1 channel 1, channel specific to ADC1. This channel is shared with ADC internal channel connected to temperature sensor, selection is done using function
LL_ADC_CHANNEL_DAC1CH2	ADC internal channel connected to DAC1 channel 2, channel specific to ADC1. This channel is shared with ADC internal channel connected to Vbat, selection is done using function

Channel - Sampling time

LL_ADC_SAMPLINGTIME_2CYCLES_5	Sampling time 2.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_6CYCLES_5	Sampling time 6.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_12CYCLES_5	Sampling time 12.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_24CYCLES_5	Sampling time 24.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_47CYCLES_5	Sampling time 47.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_92CYCLES_5	Sampling time 92.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_247CYCLES_5	Sampling time 247.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_640CYCLES_5	Sampling time 640.5 ADC clock cycles

Channel - Single or differential ending

LL_ADC_SINGLE_ENDED	ADC channel ending set to single ended (literal also used to set calibration mode)

LL_ADC_DIFFERENTIAL_ENDED	ADC channel ending set to differential (literal also used to set calibration mode)
LL_ADC_BOTH_SINGLE_DIFF_ENDED	ADC channel ending set to both single ended and differential (literal used only to set calibration factors)

ADC common - Clock source

LL_ADC_CLOCK_SYNC_PCLK_DIV1	ADC synchronous clock derived from AHB clock without prescaler
LL_ADC_CLOCK_SYNC_PCLK_DIV2	ADC synchronous clock derived from AHB clock with prescaler division by 2
LL_ADC_CLOCK_SYNC_PCLK_DIV4	ADC synchronous clock derived from AHB clock with prescaler division by 4
LL_ADC_CLOCK_ASYNC_DIV1	ADC asynchronous clock without prescaler
LL_ADC_CLOCK_ASYNC_DIV2	ADC asynchronous clock with prescaler division by 2
LL_ADC_CLOCK_ASYNC_DIV4	ADC asynchronous clock with prescaler division by 4
LL_ADC_CLOCK_ASYNC_DIV6	ADC asynchronous clock with prescaler division by 6
LL_ADC_CLOCK_ASYNC_DIV8	ADC asynchronous clock with prescaler division by 8
LL_ADC_CLOCK_ASYNC_DIV10	ADC asynchronous clock with prescaler division by 10
LL_ADC_CLOCK_ASYNC_DIV12	ADC asynchronous clock with prescaler division by 12
LL_ADC_CLOCK_ASYNC_DIV16	ADC asynchronous clock with prescaler division by 16
LL_ADC_CLOCK_ASYNC_DIV32	ADC asynchronous clock with prescaler division by 32
LL_ADC_CLOCK_ASYNC_DIV64	ADC asynchronous clock with prescaler division by 64
LL_ADC_CLOCK_ASYNC_DIV128	ADC asynchronous clock with prescaler division by 128
LL_ADC_CLOCK_ASYNC_DIV256	ADC asynchronous clock with prescaler division by 256

ADC common - Measurement path to internal channels

LL_ADC_PATH_INTERNAL_NONE	ADC measurement pathes all disabled
LL_ADC_PATH_INTERNAL_VREFINT	ADC measurement path to internal channel VrefInt
LL_ADC_PATH_INTERNAL_TEMPSENSOR	ADC measurement path to internal channel temperature sensor
LL_ADC_PATH_INTERNAL_VBAT	ADC measurement path to internal channel Vbat

ADC instance - Data alignment

LL_ADC_DATA_ALIGN_RIGHT	ADC conversion data alignment: right aligned (alignment on data register LSB bit 0)
LL_ADC_DATA_ALIGN_LEFT	ADC conversion data alignment: left aligned (aligment on data register MSB bit 15)

ADC flags

LL_ADC_FLAG_ADRDY	ADC flag ADC instance ready
LL_ADC_FLAG_EOC	ADC flag ADC group regular end of unitary conversion
LL_ADC_FLAG_EOS	ADC flag ADC group regular end of sequence conversions
LL_ADC_FLAG_OVR	ADC flag ADC group regular overrun
LL_ADC_FLAG_EOSMP	ADC flag ADC group regular end of sampling phase
LL_ADC_FLAG_JEOC	ADC flag ADC group injected end of unitary conversion
LL_ADC_FLAG_JEOS	ADC flag ADC group injected end of sequence conversions
LL_ADC_FLAG_JQOVF	ADC flag ADC group injected contexts queue overflow
LL_ADC_FLAG_AWD1	ADC flag ADC analog watchdog 1
LL_ADC_FLAG_AWD2	ADC flag ADC analog watchdog 2
LL_ADC_FLAG_AWD3	ADC flag ADC analog watchdog 3

ADC instance - Groups

LL_ADC_GROUP_REGULAR	ADC group regular (available on all STM32 devices)
LL_ADC_GROUP_INJECTED	ADC group injected (not available on all STM32 devices)
LL_ADC_GROUP_REGULAR_INJECTED	ADC both groups regular and injected

Definitions of ADC hardware constraints delays

LL_ADC_DELAY_INTERNAL_REGUL_STAB_US	Delay for ADC stabilization time (ADC voltage regulator start-up time)
LL_ADC_DELAY_VREFINT_STAB_US	Delay for internal voltage reference stabilization time
LL_ADC_DELAY_TEMPSENSOR_STAB_US	Delay for temperature sensor stabilization time
LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES	Delay required between ADC end of calibration and ADC enable

ADC group injected - Context queue mode

LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE

LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

LL_ADC_INJ_QUEUE_DISABLE

ADC group injected - Sequencer discontinuous mode

LL_ADC_INJ_SEQ_DISCONT_DISABLE	ADC group injected sequencer discontinuous mode disable
LL_ADC_INJ_SEQ_DISCONT_1RANK	ADC group injected sequencer discontinuous mode enable with sequence interruption every

rank

ADC group injected - Sequencer ranks

LL_ADC_INJ_RANK_1	ADC group injected sequencer rank 1
LL_ADC_INJ_RANK_2	ADC group injected sequencer rank 2
LL_ADC_INJ_RANK_3	ADC group injected sequencer rank 3
LL_ADC_INJ_RANK_4	ADC group injected sequencer rank 4

ADC group injected - Sequencer scan length

LL_ADC_INJ_SEQ_SCAN_DISABLE	ADC group injected sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)
LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS	ADC group injected sequencer enable with 2 ranks in the sequence
LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS	ADC group injected sequencer enable with 3 ranks in the sequence
LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS	ADC group injected sequencer enable with 4 ranks in the sequence

ADC group injected - Trigger edge

LL_ADC_INJ_TRIG_EXT_RISING	ADC group injected conversion trigger polarity set to rising edge
LL_ADC_INJ_TRIG_EXT_FALLING	ADC group injected conversion trigger polarity set to falling edge
LL_ADC_INJ_TRIG_EXT_RISINGFALLING	ADC group injected conversion trigger polarity set to both rising and falling edges

ADC group injected - Trigger source

LL_ADC_INJ_TRIG_SOFTWARE	ADC group injected conversion trigger internal: SW start.. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM1_TRGO	ADC group injected conversion trigger from external IP: TIM1 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2	ADC group injected conversion trigger from external IP: TIM1 TRGO2. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM1_CH4	ADC group injected conversion trigger from external IP: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM2_TRGO	ADC group injected conversion trigger from external IP: TIM2 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM2_CH1	ADC group injected conversion trigger from external IP: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_TRGO	ADC group injected conversion trigger from external IP: TIM3 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM3_CH1	ADC group injected conversion trigger from external IP: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM3_CH3	ADC group injected conversion trigger from external IP: TIM3 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM3_CH4	ADC group injected conversion trigger from external IP: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM4_TRGO	ADC group injected conversion trigger from external IP: TIM4 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM6_TRGO	ADC group injected conversion trigger from external IP: TIM6 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM8_CH4	ADC group injected conversion trigger from external IP: TIM8 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM8_TRGO	ADC group injected conversion trigger from external IP: TIM8 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2	ADC group injected conversion trigger from external IP: TIM8 TRGO2. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM15_TRGO	ADC group injected conversion trigger from external IP: TIM15 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_EXTI_LINE15	ADC group injected conversion trigger from external IP: external interrupt line 15. Trigger edge set to rising edge (default setting).

ADC group injected - Automatic trigger mode

LL_ADC_INJ_TRIG_INDEPENDENT	ADC group injected conversion trigger independent. Setting mandatory if ADC group injected injected trigger source is set to an external trigger.
LL_ADC_INJ_TRIG_FROM_GRP_REGULAR	ADC group injected conversion trigger from ADC group regular. Setting compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is

controlled only from ADC group regular.

ADC interruptions for configuration (interruption enable or disable)

LL_ADC_IT_ADRDY	ADC interruption ADC instance ready
LL_ADC_IT_EOC	ADC interruption ADC group regular end of unitary conversion
LL_ADC_IT_EOS	ADC interruption ADC group regular end of sequence conversions
LL_ADC_IT_OVR	ADC interruption ADC group regular overrun
LL_ADC_IT_EOSMP	ADC interruption ADC group regular end of sampling phase
LL_ADC_IT_JEOC	ADC interruption ADC group injected end of unitary conversion
LL_ADC_IT_JEOS	ADC interruption ADC group injected end of sequence conversions
LL_ADC_IT_JQOVF	ADC interruption ADC group injected contexts queue overflow
LL_ADC_IT_AWD1	ADC interruption ADC analog watchdog 1
LL_ADC_IT_AWD2	ADC interruption ADC analog watchdog 2
LL_ADC_IT_AWD3	ADC interruption ADC analog watchdog 3

ADC literals legacy naming

LL_ADC_REG_TRIG_SW_START

LL_ADC_REG_TRIG_EXT_TIM1_CC1

LL_ADC_REG_TRIG_EXT_TIM1_CC2

LL_ADC_REG_TRIG_EXT_TIM1_CC3

LL_ADC_REG_TRIG_EXT_TIM2_CC2

LL_ADC_REG_TRIG_EXT_TIM3_CC4

LL_ADC_REG_TRIG_EXT_TIM4_CC4

LL_ADC_INJ_TRIG_SW_START

LL_ADC_INJ_TRIG_EXT_TIM1_CC4

LL_ADC_INJ_TRIG_EXT_TIM2_CC1

LL_ADC_INJ_TRIG_EXT_TIM3_CC1

LL_ADC_INJ_TRIG_EXT_TIM3_CC3

LL_ADC_INJ_TRIG_EXT_TIM3_CC4

LL_ADC_INJ_TRIG_EXT_TIM8_CC4

LL_ADC_OVS_DATA_SHIFT_NONE

LL_ADC_OVS_DATA_SHIFT_1

LL_ADC_OVS_DATA_SHIFT_2

LL_ADC_OVS_DATA_SHIFT_3

LL_ADC_OVS_DATA_SHIFT_4

LL_ADC_OVS_DATA_SHIFT_5

LL_ADC_OVS_DATA_SHIFT_6

LL_ADC_OVS_DATA_SHIFT_7

LL_ADC_OVS_DATA_SHIFT_8

ADC instance - Low power mode

LL_ADC_LP_MODE_NONE	No ADC low power mode activated
LL_ADC_LP_AUTOWAIT	ADC low power mode auto delay: Dynamic low power mode, ADC conversions are performed only when necessary (when previous ADC conversion data is read). See description with function

ADC instance - Offset number

LL_ADC_OFFSET_1	ADC offset number 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)
LL_ADC_OFFSET_2	ADC offset number 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)
LL_ADC_OFFSET_3	ADC offset number 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)
LL_ADC_OFFSET_4	ADC offset number 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC instance - Offset state

LL_ADC_OFFSET_DISABLE	ADC offset disabled (among ADC selected offset number 1, 2, 3 or 4)
LL_ADC_OFFSET_ENABLE	ADC offset enabled (among ADC selected offset number 1, 2, 3 or 4)

Oversampling - Discontinuous mode

LL_ADC_OVS_REG_CONT	ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)
LL_ADC_OVS_REG_DISCONT	ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)

Oversampling - Ratio

LL_ADC_OVS_RATIO_2	ADC oversampling ratio of 2 (2 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_4	ADC oversampling ratio of 4 (4 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_8	ADC oversampling ratio of 8 (8 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_16	ADC oversampling ratio of 16 (16 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_32	ADC oversampling ratio of 32 (32 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_64	ADC oversampling ratio of 64 (64 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_128	ADC oversampling ratio of 128 (128 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_256	ADC oversampling ratio of 256 (256 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

Oversampling - Oversampling scope

LL_ADC_OVS_DISABLE	ADC oversampling disabled.
LL_ADC_OVS_GRP_REGULAR_CONTINUED	ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is temporary stopped and continued afterwards.
LL_ADC_OVS_GRP_REGULAR_RESUMED	ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).
LL_ADC_OVS_GRP_INJECTED	ADC oversampling on conversions of ADC group injected.
LL_ADC_OVS_GRP_INJ_REG_RESUMED	ADC oversampling on conversions of both ADC groups regular and injected. If group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

Oversampling - Data shift

LL_ADC_OVS_SHIFT_NONE	ADC oversampling no shift (sum of the ADC conversions data is not divided to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_1	ADC oversampling shift of 1 (sum of the ADC conversions data is divided by 2 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_2	ADC oversampling shift of 2 (sum of the ADC conversions data is divided by 4 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_3	ADC oversampling shift of 3 (sum of the ADC conversions data is divided by 8 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_4	ADC oversampling shift of 4 (sum of the ADC conversions data is divided by 16 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_5	ADC oversampling shift of 5 (sum of the ADC conversions data is divided by 32 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_6	ADC oversampling shift of 6 (sum of the ADC conversions data is divided by 64 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_7	ADC oversampling shift of 7 (sum of the ADC conversions data is divided by 128 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_8	ADC oversampling shift of 8 (sum of the ADC conversions data is divided by 256 to result as the ADC oversampling conversion data)

ADC registers compliant with specific purpose

LL_ADC_DMA_REG_REGULAR_DATA

ADC group regular - Continuous mode

LL_ADC_REG_CONV_SINGLE	ADC conversions are performed in single mode: one conversion per trigger
LL_ADC_REG_CONV_CONTINUOUS	ADC conversions are performed in continuous mode: after the first trigger, following conversions launched successively automatically

ADC group regular - DFSDM transfer of ADC conversion data

LL_ADC_REG_DFSDM_TRANSFER_NONE	ADC conversions are not transferred by DFSDM.
LL_ADC_REG_DFSDM_TRANSFER_ENABLE	ADC conversion data are transfered to DFSDM for post processing. The ADC conversion data format must be 16-bit signed and right aligned, refer to reference manual. DFSDM transfer cannot be used if DMA transfer is enabled.

ADC group regular - DMA transfer of ADC conversion data

LL_ADC_REG_DMA_TRANSFER_NONE	ADC conversions are not transferred by DMA
LL_ADC_REG_DMA_TRANSFER_LIMITED	ADC conversion data are transferred by

	DMA, in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.
LL_ADC_REG_DMA_TRANSFER_UNLIMITED	ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

ADC group regular - Overrun behavior on conversion data

LL_ADC_REG_OVR_DATA_PRESERVED	ADC group regular behavior in case of overrun: data preserved
LL_ADC_REG_OVR_DATA_OVERWRITTEN	ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sequencer discontinuous mode

LL_ADC_REG_SEQ_DISCONT_DISABLE	ADC group regular sequencer discontinuous mode disable
LL_ADC_REG_SEQ_DISCONT_1RANK	ADC group regular sequencer discontinuous mode enable with sequence interruption every rank
LL_ADC_REG_SEQ_DISCONT_2RANKS	ADC group regular sequencer discontinuous mode enabled with sequence interruption every 2 ranks
LL_ADC_REG_SEQ_DISCONT_3RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 3 ranks
LL_ADC_REG_SEQ_DISCONT_4RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 4 ranks
LL_ADC_REG_SEQ_DISCONT_5RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 5 ranks
LL_ADC_REG_SEQ_DISCONT_6RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 6 ranks
LL_ADC_REG_SEQ_DISCONT_7RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 7 ranks
LL_ADC_REG_SEQ_DISCONT_8RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 8 ranks

ADC group regular - Sequencer ranks

LL_ADC_REG_RANK_1	ADC group regular sequencer rank 1

LL_ADC_REG_RANK_2	ADC group regular sequencer rank 2
LL_ADC_REG_RANK_3	ADC group regular sequencer rank 3
LL_ADC_REG_RANK_4	ADC group regular sequencer rank 4
LL_ADC_REG_RANK_5	ADC group regular sequencer rank 5
LL_ADC_REG_RANK_6	ADC group regular sequencer rank 6
LL_ADC_REG_RANK_7	ADC group regular sequencer rank 7
LL_ADC_REG_RANK_8	ADC group regular sequencer rank 8
LL_ADC_REG_RANK_9	ADC group regular sequencer rank 9
LL_ADC_REG_RANK_10	ADC group regular sequencer rank 10
LL_ADC_REG_RANK_11	ADC group regular sequencer rank 11
LL_ADC_REG_RANK_12	ADC group regular sequencer rank 12
LL_ADC_REG_RANK_13	ADC group regular sequencer rank 13
LL_ADC_REG_RANK_14	ADC group regular sequencer rank 14
LL_ADC_REG_RANK_15	ADC group regular sequencer rank 15
LL_ADC_REG_RANK_16	ADC group regular sequencer rank 16

ADC group regular - Sequencer scan length

LL_ADC_REG_SEQ_SCAN_DISABLE	ADC group regular sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)
LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS	ADC group regular sequencer enable with 2 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS	ADC group regular sequencer enable with 3 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS	ADC group regular sequencer enable with 4 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS	ADC group regular sequencer enable with 5 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS	ADC group regular sequencer enable with 6 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS	ADC group regular sequencer enable with 7 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS	ADC group regular sequencer enable with 8 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS	ADC group regular sequencer enable with 9 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS	ADC group regular sequencer enable with 10 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS	ADC group regular sequencer enable with 11 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS	ADC group regular sequencer enable with 12 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS	ADC group regular sequencer enable with 13 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS	ADC group regular sequencer enable with 14 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS	ADC group regular sequencer enable with 15 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS	ADC group regular sequencer enable with 16 ranks in the sequence

ADC group regular - Trigger edge

LL_ADC_REG_TRIG_EXT_RISING	ADC group regular conversion trigger polarity set to rising edge
LL_ADC_REG_TRIG_EXT_FALLING	ADC group regular conversion trigger polarity set to falling edge
LL_ADC_REG_TRIG_EXT_RISINGFALLING	ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular - Trigger source

LL_ADC_REG_TRIG_SOFTWARE	ADC group regular conversion trigger internal: SW start.
LL_ADC_REG_TRIG_EXT_TIM1_TRGO	ADC group regular conversion trigger from external IP: TIM1 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM1_TRGO2	ADC group regular conversion trigger from external IP: TIM1 TRGO2. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM1_CH1	ADC group regular conversion trigger from external IP: TIM1 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM1_CH2	ADC group regular conversion trigger from external IP: TIM1 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM1_CH3	ADC group regular conversion trigger from external IP: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM2_TRGO	ADC group regular conversion trigger from external IP: TIM2 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM2_CH2	ADC group regular conversion trigger from external IP: TIM2 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM3_TRGO	ADC group regular conversion trigger from external IP: TIM3 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM3_CH4	ADC group regular conversion trigger from external IP: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM4_TRGO	ADC group regular conversion trigger from external IP: TIM4 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM4_CH4	ADC group regular conversion trigger from external IP: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM6_TRGO	ADC group regular conversion trigger from external IP: TIM6 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM8_TRGO	ADC group regular conversion trigger from external IP: TIM8 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM8_TRGO2	ADC group regular conversion trigger from external IP: TIM8 TRGO2. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM15_TRGO	ADC group regular conversion trigger from external IP: TIM15 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_EXTI_LINE11	ADC group regular conversion trigger from external IP: external interrupt line 11. Trigger edge set to rising edge (default setting).

ADC instance - Resolution

LL_ADC_RESOLUTION_12B	ADC resolution 12 bits
LL_ADC_RESOLUTION_10B	ADC resolution 10 bits
LL_ADC_RESOLUTION_8B	ADC resolution 8 bits
LL_ADC_RESOLUTION_6B	ADC resolution 6 bits

ADC instance - ADC sampling time common configuration

LL_ADC_SAMPLINGTIME_COMMON_DEFAULT	ADC sampling time let to default settings.
LL_ADC_SAMPLINGTIME_COMMON_3C5_REPL_2C5	ADC additional sampling time 3.5 ADC clock cycles replacing 2.5 ADC clock cycles (this applies to all channels mapped with selection sampling time 2.5 ADC clock cycles, whatever channels mapped on ADC groups regular or injected).

ADC helper macro

__LL_ADC_CHANNEL_TO_DECIMAL_NB	**Description:** • Helper macro to get ADC channel number in decimal format from literals LL_ADC_CHANNEL_x. **Parameters:** • __CHANNEL__: This parameter can be one of the following values: – LL_ADC_CHANNEL_0 – LL_ADC_CHANNEL_1 (7) – LL_ADC_CHANNEL_2 (7) – LL_ADC_CHANNEL_3 (7) – LL_ADC_CHANNEL_4 (7) – LL_ADC_CHANNEL_5 (7) – LL_ADC_CHANNEL_6 – LL_ADC_CHANNEL_7 – LL_ADC_CHANNEL_8 – LL_ADC_CHANNEL_9 – LL_ADC_CHANNEL_10 – LL_ADC_CHANNEL_11 – LL_ADC_CHANNEL_12 – LL_ADC_CHANNEL_13 – LL_ADC_CHANNEL_14 – LL_ADC_CHANNEL_15 – LL_ADC_CHANNEL_16 – LL_ADC_CHANNEL_17 – LL_ADC_CHANNEL_18 – LL_ADC_CHANNEL_VREFINT (1) – LL_ADC_CHANNEL_TEMPSENSOR (4) – LL_ADC_CHANNEL_VBAT (4) – LL_ADC_CHANNEL_DAC1CH1 (5) – LL_ADC_CHANNEL_DAC1CH2 (5) – LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6) – LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6) – LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6) – LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6) **Return value:** • Value: between Min_Data=0 and Max_Data=18 **Notes:** • Example: __LL_ADC_CHANNEL_TO_DECIMAL_NB(LL_ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).
__LL_ADC_DECIMAL_NB_TO_CHANNEL	**Description:** • Helper macro to get ADC channel in literal format LL_ADC_CHANNEL_x from number in decimal

format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (7)
  - LL_ADC_CHANNEL_2 (7)
  - LL_ADC_CHANNEL_3 (7)
  - LL_ADC_CHANNEL_4 (7)
  - LL_ADC_CHANNEL_5 (7)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (4)
  - LL_ADC_CHANNEL_VBAT (4)
  - LL_ADC_CHANNEL_DAC1CH1 (5)
  - LL_ADC_CHANNEL_DAC1CH2 (5)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

Notes:

- Example: __LL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "LL_ADC_CHANNEL_4".

__LL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (7)

- LL_ADC_CHANNEL_2 (7)
- LL_ADC_CHANNEL_3 (7)
- LL_ADC_CHANNEL_4 (7)
- LL_ADC_CHANNEL_5 (7)
- LL_ADC_CHANNEL_6
- LL_ADC_CHANNEL_7
- LL_ADC_CHANNEL_8
- LL_ADC_CHANNEL_9
- LL_ADC_CHANNEL_10
- LL_ADC_CHANNEL_11
- LL_ADC_CHANNEL_12
- LL_ADC_CHANNEL_13
- LL_ADC_CHANNEL_14
- LL_ADC_CHANNEL_15
- LL_ADC_CHANNEL_16
- LL_ADC_CHANNEL_17
- LL_ADC_CHANNEL_18
- LL_ADC_CHANNEL_VREFINT (1)
- LL_ADC_CHANNEL_TEMPSENSOR (4)
- LL_ADC_CHANNEL_VBAT (4)
- LL_ADC_CHANNEL_DAC1CH1 (5)
- LL_ADC_CHANNEL_DAC1CH2 (5)
- LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
- LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
- LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
- LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (7)
  - LL_ADC_CHANNEL_2 (7)
  - LL_ADC_CHANNEL_3 (7)
  - LL_ADC_CHANNEL_4 (7)
  - LL_ADC_CHANNEL_5 (7)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (4)
  - LL_ADC_CHANNEL_VBAT (4)
  - LL_ADC_CHANNEL_DAC1CH1 (5)
  - LL_ADC_CHANNEL_DAC1CH2 (5)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7

- – LL_ADC_CHANNEL_8
- – LL_ADC_CHANNEL_9
- – LL_ADC_CHANNEL_10
- – LL_ADC_CHANNEL_11
- – LL_ADC_CHANNEL_12
- – LL_ADC_CHANNEL_13
- – LL_ADC_CHANNEL_14
- – LL_ADC_CHANNEL_15
- – LL_ADC_CHANNEL_16
- – LL_ADC_CHANNEL_17
- – LL_ADC_CHANNEL_18

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__LL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (4)
  - LL_ADC_CHANNEL_VBAT (4)
  - LL_ADC_CHANNEL_DAC1CH1 (5)
  - LL_ADC_CHANNEL_DAC1CH2 (5)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of

ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_ANALOGWD_CHANNEL_GROUP

Description:

- Helper macro to define ADC analog watchdog parameter: define a single channel to monitor with analog watchdog from sequencer channel and groups definition.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (7)
  - LL_ADC_CHANNEL_2 (7)
  - LL_ADC_CHANNEL_3 (7)
  - LL_ADC_CHANNEL_4 (7)
  - LL_ADC_CHANNEL_5 (7)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (4)
  - LL_ADC_CHANNEL_VBAT (4)
  - LL_ADC_CHANNEL_DAC1CH1 (5)
  - LL_ADC_CHANNEL_DAC1CH2 (5)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)(6)
  - LL_ADC_CHANNEL_DAC1CH1_ADC3 (3)(6)
  - LL_ADC_CHANNEL_DAC1CH2_ADC3 (3)(6)
- __GROUP__: This parameter can be one of the following values:
  - LL_ADC_GROUP_REGULAR
  - LL_ADC_GROUP_INJECTED
  - LL_ADC_GROUP_REGULAR_INJECTED

Return value:

- Returned: value can be one of the following

values:

- LL_ADC_AWD_DISABLE
- LL_ADC_AWD_ALL_CHANNELS_REG (0)
- LL_ADC_AWD_ALL_CHANNELS_INJ (0)
- LL_ADC_AWD_ALL_CHANNELS_REG_INJ
- LL_ADC_AWD_CHANNEL_0_REG (0)
- LL_ADC_AWD_CHANNEL_0_INJ (0)
- LL_ADC_AWD_CHANNEL_0_REG_INJ
- LL_ADC_AWD_CHANNEL_1_REG (0)
- LL_ADC_AWD_CHANNEL_1_INJ (0)
- LL_ADC_AWD_CHANNEL_1_REG_INJ
- LL_ADC_AWD_CHANNEL_2_REG (0)
- LL_ADC_AWD_CHANNEL_2_INJ (0)
- LL_ADC_AWD_CHANNEL_2_REG_INJ
- LL_ADC_AWD_CHANNEL_3_REG (0)
- LL_ADC_AWD_CHANNEL_3_INJ (0)
- LL_ADC_AWD_CHANNEL_3_REG_INJ
- LL_ADC_AWD_CHANNEL_4_REG (0)
- LL_ADC_AWD_CHANNEL_4_INJ (0)
- LL_ADC_AWD_CHANNEL_4_REG_INJ
- LL_ADC_AWD_CHANNEL_5_REG (0)
- LL_ADC_AWD_CHANNEL_5_INJ (0)
- LL_ADC_AWD_CHANNEL_5_REG_INJ
- LL_ADC_AWD_CHANNEL_6_REG (0)
- LL_ADC_AWD_CHANNEL_6_INJ (0)
- LL_ADC_AWD_CHANNEL_6_REG_INJ
- LL_ADC_AWD_CHANNEL_7_REG (0)
- LL_ADC_AWD_CHANNEL_7_INJ (0)
- LL_ADC_AWD_CHANNEL_7_REG_INJ
- LL_ADC_AWD_CHANNEL_8_REG (0)
- LL_ADC_AWD_CHANNEL_8_INJ (0)
- LL_ADC_AWD_CHANNEL_8_REG_INJ
- LL_ADC_AWD_CHANNEL_9_REG (0)
- LL_ADC_AWD_CHANNEL_9_INJ (0)
- LL_ADC_AWD_CHANNEL_9_REG_INJ
- LL_ADC_AWD_CHANNEL_10_REG (0)
- LL_ADC_AWD_CHANNEL_10_INJ (0)
- LL_ADC_AWD_CHANNEL_10_REG_INJ
- LL_ADC_AWD_CHANNEL_11_REG (0)
- LL_ADC_AWD_CHANNEL_11_INJ (0)
- LL_ADC_AWD_CHANNEL_11_REG_INJ
- LL_ADC_AWD_CHANNEL_12_REG (0)
- LL_ADC_AWD_CHANNEL_12_INJ (0)
- LL_ADC_AWD_CHANNEL_12_REG_INJ
- LL_ADC_AWD_CHANNEL_13_REG (0)
- LL_ADC_AWD_CHANNEL_13_INJ (0)
- LL_ADC_AWD_CHANNEL_13_REG_INJ
- LL_ADC_AWD_CHANNEL_14_REG (0)
- LL_ADC_AWD_CHANNEL_14_INJ (0)
- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG (0)
- LL_ADC_AWD_CHANNEL_15_INJ (0)

- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG (0)
- LL_ADC_AWD_CHANNEL_16_INJ (0)
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG (0)
- LL_ADC_AWD_CHANNEL_17_INJ (0)
- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CHANNEL_18_REG (0)
- LL_ADC_AWD_CHANNEL_18_INJ (0)
- LL_ADC_AWD_CHANNEL_18_REG_INJ
- LL_ADC_AWD_CH_VREFINT_REG (0)(1)
- LL_ADC_AWD_CH_VREFINT_INJ (0)(1)
- LL_ADC_AWD_CH_VREFINT_REG_INJ (1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG (0)(4)
- LL_ADC_AWD_CH_TEMPSENSOR_INJ (0)(4)
- LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ (4)
- LL_ADC_AWD_CH_VBAT_REG (0)(4)
- LL_ADC_AWD_CH_VBAT_INJ (0)(4)
- LL_ADC_AWD_CH_VBAT_REG_INJ (4)
- LL_ADC_AWD_CH_DAC1CH1_REG (0)(2)(5)
- LL_ADC_AWD_CH_DAC1CH1_INJ (0)(2)(5)
- LL_ADC_AWD_CH_DAC1CH1_REG_INJ (2)(5)
- LL_ADC_AWD_CH_DAC1CH2_REG (0)(2)(5)
- LL_ADC_AWD_CH_DAC1CH2_INJ (0)(2)(5)
- LL_ADC_AWD_CH_DAC1CH2_REG_INJ (2)(5)
- LL_ADC_AWD_CH_DAC1CH1_ADC2_REG (0)(2)(6)
- LL_ADC_AWD_CH_DAC1CH1_ADC2_INJ (0)(2)(6)
- LL_ADC_AWD_CH_DAC1CH1_ADC2_REG_INJ (2)(6)
- LL_ADC_AWD_CH_DAC1CH2_ADC2_REG (0)(2)(6)
- LL_ADC_AWD_CH_DAC1CH2_ADC2_INJ (0)(2)(6)
- LL_ADC_AWD_CH_DAC1CH2_ADC2_REG_INJ (2)(6)
- LL_ADC_AWD_CH_DAC1CH1_ADC3_REG (0)(3)(6)
- LL_ADC_AWD_CH_DAC1CH1_ADC3_INJ (0)(3)(6)
- LL_ADC_AWD_CH_DAC1CH1_ADC3_REG_INJ (3)(6)
- LL_ADC_AWD_CH_DAC1CH2_ADC3_REG (0)(3)(6)
- LL_ADC_AWD_CH_DAC1CH2_ADC3_INJ

(0)(3)(6)
- LL_ADC_AWD_CH_DAC1CH2_ADC3_REG_INJ (3)(6)

Notes:

- To be used with function LL_ADC_SetAnalogWDMonitChannels(). Example: LL_ADC_SetAnalogWDMonitChannels(ADC1, LL_ADC_AWD1, __LL_ADC_ANALOGWD_CHANNEL_GROUP(LL_ADC_CHANNEL4, LL_ADC_GROUP_REGULAR))

__LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION

Description:

- Helper macro to set the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __AWD_THRESHOLD__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_ConfigAnalogWDThresholds() or LL_ADC_SetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to set the value of analog watchdog threshold high (on 8 bits): LL_ADC_SetAnalogWDThresholds (< ADCx param>, __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(LL_ADC_RESOLUTION_8B, <threshold_value_8_bits>));

__LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION

Description:

- Helper macro to get the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:

  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __AWD_THRESHOLD_12_BITS__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to get the value of analog watchdog threshold high (on 8 bits): < threshold_value_6_bits> = __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION (LL_ADC_RESOLUTION_8B, LL_ADC_GetAnalogWDThresholds(<ADCx param>, LL_ADC_AWD_THRESHOLD_HIGH));

__LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW

Description:

- Helper macro to get the ADC analog watchdog threshold high or low from raw value containing both thresholds concatenated.

Parameters:

- __AWD_THRESHOLD_TYPE__: This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
- __AWD_THRESHOLDS__: Value between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, to get analog watchdog threshold high from the register raw value: __LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW(LL_ADC_AWD_THRESHOLD_HIGH, <raw_value_with_both_thresholds>);

__LL_ADC_CALIB_FACTOR_SINGLE_DIFF

Description:

- Helper macro to set the ADC calibration value with both single ended and differential modes calibration factors concatenated.

Parameters:

- __CALIB_FACTOR_SINGLE_ENDED__: Value between Min_Data=0x00 and Max_Data=0x7F
- __CALIB_FACTOR_DIFFERENTIAL__: Value between Min_Data=0x00 and Max_Data=0x7F

Return value:

- Value: between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes:

- To be used with function LL_ADC_SetCalibrationFactor(). Example, to set calibration factors single ended to 0x55 and differential ended to 0x2A: LL_ADC_SetCalibrationFactor(ADC1, __LL_ADC_CALIB_FACTOR_SINGLE_DIFF(0x55, 0x2A))

__LL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__LL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of to the data to be converted This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__LL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__LL_ADC_CALC_DATA_VOLTAGE

__LL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

Parameters:

- __VREFINT_ADC_DATA__: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 serie, calibration data of internal voltage reference VrefInt corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__LL_ADC_CALC_TEMPERATURE

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS_ADC_DATA - TS_CAL1) * (TS_CAL2_TEMP - TS_CAL1_TEMP)) / (TS_CAL2 - TS_CAL1) + TS_CAL1_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC Avg_Slope = (TS_CAL2 - TS_CAL1) / (TS_CAL2_TEMP - TS_CAL1_TEMP) TS_CAL1 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) TS_CAL2 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). On this STM32 serie, calibration data of temperature sensor corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__LL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32L4, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On STM32L4, refer to device datasheet parameter "V30" (corresponding to TS_CAL1).
- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). ADC measurement data must correspond to a resolution of 12bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 12 bits.

Common write and read registers Macros

LL_ADC_WriteReg

Description:

- Write a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_ADC_ReadReg

Description:

- Read a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be read

Return value:

- Register: value

74 LL BUS Generic Driver

74.1 BUS Firmware driver API description

74.1.1 Detailed description of functions

LL_AHB1_GRP1_EnableClock

Function name	**__STATIC_INLINE void LL_AHB1_GRP1_EnableClock (uint32_t Periphs)**
Function description	Enable AHB1 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB1_GRP1_PERIPH_DMA1 – LL_AHB1_GRP1_PERIPH_DMA2 – LL_AHB1_GRP1_PERIPH_DMAMUX1 (*) – LL_AHB1_GRP1_PERIPH_FLASH – LL_AHB1_GRP1_PERIPH_CRC – LL_AHB1_GRP1_PERIPH_TSC – LL_AHB1_GRP1_PERIPH_DMA2D (*) – LL_AHB1_GRP1_PERIPH_GFXMMU (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• AHB1ENR DMA1EN LL_AHB1_GRP1_EnableClock • AHB1ENR DMA2EN LL_AHB1_GRP1_EnableClock • AHB1ENR DMAMUX1EN LL_AHB1_GRP1_EnableClock • AHB1ENR FLASHEN LL_AHB1_GRP1_EnableClock • AHB1ENR CRCEN LL_AHB1_GRP1_EnableClock • AHB1ENR TSCEN LL_AHB1_GRP1_EnableClock • AHB1ENR DMA2DEN LL_AHB1_GRP1_EnableClock • AHB1ENR GFXMMUEN LL_AHB1_GRP1_EnableClock

LL_AHB1_GRP1_IsEnabledClock

Function name	**__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClock (uint32_t Periphs)**
Function description	Check if AHB1 peripheral clock is enabled or not.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB1_GRP1_PERIPH_DMA1 – LL_AHB1_GRP1_PERIPH_DMA2 – LL_AHB1_GRP1_PERIPH_DMAMUX1 (*) – LL_AHB1_GRP1_PERIPH_FLASH – LL_AHB1_GRP1_PERIPH_CRC – LL_AHB1_GRP1_PERIPH_TSC – LL_AHB1_GRP1_PERIPH_DMA2D (*) – LL_AHB1_GRP1_PERIPH_GFXMMU (*)
Return values	• **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:
- AHB1ENR DMA1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR DMA2EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR DMAMUX1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR FLASHEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR CRCEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR TSCEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR DMA2DEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR GFXMMUEN LL_AHB1_GRP1_IsEnabledClock

LL_AHB1_GRP1_DisableClock

Function name: **__STATIC_INLINE void LL_AHB1_GRP1_DisableClock (uint32_t Periphs)**

Function description: Disable AHB1 peripherals clock.

Parameters:
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_DMAMUX1 (*)
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_TSC
  - LL_AHB1_GRP1_PERIPH_DMA2D (*)
  - LL_AHB1_GRP1_PERIPH_GFXMMU (*)

Return values:
- **None:**

Reference Manual to LL API cross reference:
- AHB1ENR DMA1EN LL_AHB1_GRP1_DisableClock
- AHB1ENR DMA2EN LL_AHB1_GRP1_DisableClock
- AHB1ENR DMAMUX1EN LL_AHB1_GRP1_DisableClock
- AHB1ENR FLASHEN LL_AHB1_GRP1_DisableClock
- AHB1ENR CRCEN LL_AHB1_GRP1_DisableClock
- AHB1ENR TSCEN LL_AHB1_GRP1_DisableClock
- AHB1ENR DMA2DEN LL_AHB1_GRP1_DisableClock
- AHB1ENR GFXMMUEN LL_AHB1_GRP1_DisableClock

LL_AHB1_GRP1_ForceReset

Function name: **__STATIC_INLINE void LL_AHB1_GRP1_ForceReset (uint32_t Periphs)**

Function description: Force AHB1 peripherals reset.

Parameters:
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_DMAMUX1 (*)
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_TSC
  - LL_AHB1_GRP1_PERIPH_DMA2D (*)

– LL_AHB1_GRP1_PERIPH_GFXMMU (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- AHB1RSTR DMA1RST LL_AHB1_GRP1_ForceReset
- AHB1RSTR DMA2RST LL_AHB1_GRP1_ForceReset
- AHB1RSTR DMAMUX1RST LL_AHB1_GRP1_ForceReset
- AHB1RSTR FLASHRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR CRCRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR TSCRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR DMA2DRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR GFXMMURST LL_AHB1_GRP1_ForceReset

LL_AHB1_GRP1_ReleaseReset

Function name
__STATIC_INLINE void LL_AHB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description
Release AHB1 peripherals reset.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  – LL_AHB1_GRP1_PERIPH_ALL
  – LL_AHB1_GRP1_PERIPH_DMA1
  – LL_AHB1_GRP1_PERIPH_DMA2
  – LL_AHB1_GRP1_PERIPH_DMAMUX1 (*)
  – LL_AHB1_GRP1_PERIPH_FLASH
  – LL_AHB1_GRP1_PERIPH_CRC
  – LL_AHB1_GRP1_PERIPH_TSC
  – LL_AHB1_GRP1_PERIPH_DMA2D (*)
  – LL_AHB1_GRP1_PERIPH_GFXMMU (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- AHB1RSTR DMA1RST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR DMA2RST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR DMAMUX1RST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR FLASHRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR CRCRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR TSCRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR DMA2DRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR GFXMMURST LL_AHB1_GRP1_ReleaseReset

LL_AHB1_GRP1_EnableClockStopSleep

Function name
__STATIC_INLINE void LL_AHB1_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description
Enable AHB1 peripheral clocks in Sleep and Stop modes.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  – LL_AHB1_GRP1_PERIPH_DMA1
  – LL_AHB1_GRP1_PERIPH_DMA2
  – LL_AHB1_GRP1_PERIPH_DMAMUX1 (*)
  – LL_AHB1_GRP1_PERIPH_FLASH

- – LL_AHB1_GRP1_PERIPH_SRAM1
- – LL_AHB1_GRP1_PERIPH_CRC
- – LL_AHB1_GRP1_PERIPH_TSC
- – LL_AHB1_GRP1_PERIPH_DMA2D (*)
- – LL_AHB1_GRP1_PERIPH_GFXMMU (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- AHB1SMENR DMA1SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR DMA2SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR DMAMUX1SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR FLASHSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR SRAM1SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR CRCSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR TSCSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR DMA2DSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR GFXMMUSMEN LL_AHB1_GRP1_EnableClockStopSleep

LL_AHB1_GRP1_DisableClockStopSleep

Function name
: **__STATIC_INLINE void LL_AHB1_GRP1_DisableClockStopSleep (uint32_t Periphs)**

Function description
: Disable AHB1 peripheral clocks in Sleep and Stop modes.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - – LL_AHB1_GRP1_PERIPH_DMA1
  - – LL_AHB1_GRP1_PERIPH_DMA2
  - – LL_AHB1_GRP1_PERIPH_DMAMUX1 (*)
  - – LL_AHB1_GRP1_PERIPH_FLASH
  - – LL_AHB1_GRP1_PERIPH_SRAM1
  - – LL_AHB1_GRP1_PERIPH_CRC
  - – LL_AHB1_GRP1_PERIPH_TSC
  - – LL_AHB1_GRP1_PERIPH_DMA2D (*)
  - – LL_AHB1_GRP1_PERIPH_GFXMMU (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- AHB1SMENR DMA1SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR DMA2SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR DMAMUX1SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR FLASHSMEN LL_AHB1_GRP1_DisableClockStopSleep

- AHB1SMENR SRAM1SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR CRCSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR TSCSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR DMA2DSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR GFXMMUSMEN LL_AHB1_GRP1_DisableClockStopSleep

LL_AHB2_GRP1_EnableClock

Function name	**__STATIC_INLINE void LL_AHB2_GRP1_EnableClock (uint32_t Periphs)**
Function description	Enable AHB2 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB2_GRP1_PERIPH_GPIOA – LL_AHB2_GRP1_PERIPH_GPIOB – LL_AHB2_GRP1_PERIPH_GPIOC – LL_AHB2_GRP1_PERIPH_GPIOD (*) – LL_AHB2_GRP1_PERIPH_GPIOE (*) – LL_AHB2_GRP1_PERIPH_GPIOF (*) – LL_AHB2_GRP1_PERIPH_GPIOG (*) – LL_AHB2_GRP1_PERIPH_GPIOH – LL_AHB2_GRP1_PERIPH_GPIOI (*) – LL_AHB2_GRP1_PERIPH_OTGFS (*) – LL_AHB2_GRP1_PERIPH_ADC – LL_AHB2_GRP1_PERIPH_DCMI (*) – LL_AHB2_GRP1_PERIPH_AES (*) – LL_AHB2_GRP1_PERIPH_HASH (*) – LL_AHB2_GRP1_PERIPH_RNG – LL_AHB2_GRP1_PERIPH_OSPIM (*) – LL_AHB2_GRP1_PERIPH_SDMMC1 (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• AHB2ENR GPIOAEN LL_AHB2_GRP1_EnableClock • AHB2ENR GPIOBEN LL_AHB2_GRP1_EnableClock • AHB2ENR GPIOCEN LL_AHB2_GRP1_EnableClock • AHB2ENR GPIODEN LL_AHB2_GRP1_EnableClock • AHB2ENR GPIOEEN LL_AHB2_GRP1_EnableClock • AHB2ENR GPIOFEN LL_AHB2_GRP1_EnableClock • AHB2ENR GPIOGEN LL_AHB2_GRP1_EnableClock • AHB2ENR GPIOHEN LL_AHB2_GRP1_EnableClock • AHB2ENR GPIOIEN LL_AHB2_GRP1_EnableClock • AHB2ENR OTGFSEN LL_AHB2_GRP1_EnableClock • AHB2ENR ADCEN LL_AHB2_GRP1_EnableClock • AHB2ENR DCMIEN LL_AHB2_GRP1_EnableClock • AHB2ENR AESEN LL_AHB2_GRP1_EnableClock • AHB2ENR HASHEN LL_AHB2_GRP1_EnableClock • AHB2ENR RNGEN LL_AHB2_GRP1_EnableClock

- AHB2ENR OSPIMEN LL_AHB2_GRP1_EnableClock
- AHB2ENR SDMMC1EN LL_AHB2_GRP1_EnableClock

LL_AHB2_GRP1_IsEnabledClock

Function name	**__STATIC_INLINE uint32_t LL_AHB2_GRP1_IsEnabledClock (uint32_t Periphs)**
Function description	Check if AHB2 peripheral clock is enabled or not.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB2_GRP1_PERIPH_GPIOA – LL_AHB2_GRP1_PERIPH_GPIOB – LL_AHB2_GRP1_PERIPH_GPIOC – LL_AHB2_GRP1_PERIPH_GPIOD (*) – LL_AHB2_GRP1_PERIPH_GPIOE (*) – LL_AHB2_GRP1_PERIPH_GPIOF (*) – LL_AHB2_GRP1_PERIPH_GPIOG (*) – LL_AHB2_GRP1_PERIPH_GPIOH – LL_AHB2_GRP1_PERIPH_GPIOI (*) – LL_AHB2_GRP1_PERIPH_OTGFS (*) – LL_AHB2_GRP1_PERIPH_ADC – LL_AHB2_GRP1_PERIPH_DCMI (*) – LL_AHB2_GRP1_PERIPH_AES (*) – LL_AHB2_GRP1_PERIPH_HASH (*) – LL_AHB2_GRP1_PERIPH_RNG – LL_AHB2_GRP1_PERIPH_OSPIM (*) – LL_AHB2_GRP1_PERIPH_SDMMC1 (*)
Return values	• **State:** of Periphs (1 or 0).
Reference Manual to LL API cross reference:	• AHB2ENR GPIOAEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR GPIOBEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR GPIOCEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR GPIODEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR GPIOEEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR GPIOFEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR GPIOGEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR GPIOHEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR GPIOIEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR OTGFSEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR ADCEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR DCMIEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR AESEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR HASHEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR RNGEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR OSPIMEN LL_AHB2_GRP1_IsEnabledClock • AHB2ENR SDMMC1EN LL_AHB2_GRP1_IsEnabledClock

LL_AHB2_GRP1_DisableClock

Function name	**__STATIC_INLINE void LL_AHB2_GRP1_DisableClock (uint32_t Periphs)**

Function description	Disable AHB2 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB2_GRP1_PERIPH_GPIOA – LL_AHB2_GRP1_PERIPH_GPIOB – LL_AHB2_GRP1_PERIPH_GPIOC – LL_AHB2_GRP1_PERIPH_GPIOD (*) – LL_AHB2_GRP1_PERIPH_GPIOE (*) – LL_AHB2_GRP1_PERIPH_GPIOF (*) – LL_AHB2_GRP1_PERIPH_GPIOG (*) – LL_AHB2_GRP1_PERIPH_GPIOH – LL_AHB2_GRP1_PERIPH_GPIOI (*) – LL_AHB2_GRP1_PERIPH_OTGFS (*) – LL_AHB2_GRP1_PERIPH_ADC – LL_AHB2_GRP1_PERIPH_DCMI (*) – LL_AHB2_GRP1_PERIPH_AES (*) – LL_AHB2_GRP1_PERIPH_HASH (*) – LL_AHB2_GRP1_PERIPH_RNG – LL_AHB2_GRP1_PERIPH_OSPIM (*) – LL_AHB2_GRP1_PERIPH_SDMMC1 (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• AHB2ENR GPIOAEN LL_AHB2_GRP1_DisableClock • AHB2ENR GPIOBEN LL_AHB2_GRP1_DisableClock • AHB2ENR GPIOCEN LL_AHB2_GRP1_DisableClock • AHB2ENR GPIODEN LL_AHB2_GRP1_DisableClock • AHB2ENR GPIOEEN LL_AHB2_GRP1_DisableClock • AHB2ENR GPIOFEN LL_AHB2_GRP1_DisableClock • AHB2ENR GPIOGEN LL_AHB2_GRP1_DisableClock • AHB2ENR GPIOHEN LL_AHB2_GRP1_DisableClock • AHB2ENR GPIOIEN LL_AHB2_GRP1_DisableClock • AHB2ENR OTGFSEN LL_AHB2_GRP1_DisableClock • AHB2ENR ADCEN LL_AHB2_GRP1_DisableClock • AHB2ENR DCMIEN LL_AHB2_GRP1_DisableClock • AHB2ENR AESEN LL_AHB2_GRP1_DisableClock • AHB2ENR HASHEN LL_AHB2_GRP1_DisableClock • AHB2ENR RNGEN LL_AHB2_GRP1_DisableClock • AHB2ENR OSPIMEN LL_AHB2_GRP1_DisableClock • AHB2ENR SDMMC1EN LL_AHB2_GRP1_DisableClock

LL_AHB2_GRP1_ForceReset

Function name	**__STATIC_INLINE void LL_AHB2_GRP1_ForceReset (uint32_t Periphs)**
Function description	Force AHB2 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB2_GRP1_PERIPH_ALL – LL_AHB2_GRP1_PERIPH_GPIOA – LL_AHB2_GRP1_PERIPH_GPIOB – LL_AHB2_GRP1_PERIPH_GPIOC

- LL_AHB2_GRP1_PERIPH_GPIOD (*)
- LL_AHB2_GRP1_PERIPH_GPIOE (*)
- LL_AHB2_GRP1_PERIPH_GPIOF (*)
- LL_AHB2_GRP1_PERIPH_GPIOG (*)
- LL_AHB2_GRP1_PERIPH_GPIOH
- LL_AHB2_GRP1_PERIPH_GPIOI (*)
- LL_AHB2_GRP1_PERIPH_OTGFS (*)
- LL_AHB2_GRP1_PERIPH_ADC
- LL_AHB2_GRP1_PERIPH_DCMI (*)
- LL_AHB2_GRP1_PERIPH_AES (*)
- LL_AHB2_GRP1_PERIPH_HASH (*)
- LL_AHB2_GRP1_PERIPH_RNG
- LL_AHB2_GRP1_PERIPH_OSPIM (*)
- LL_AHB2_GRP1_PERIPH_SDMMC1 (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- AHB2RSTR GPIOARST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOBRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOCRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIODRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOERST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOFRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOGRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOHRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOIRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR OTGFSRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR ADCRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR DCMIRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR AESRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR HASHRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR RNGRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR OSPIMRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR SDMMC1RST LL_AHB2_GRP1_ForceReset

LL_AHB2_GRP1_ReleaseReset

Function name
__STATIC_INLINE void LL_AHB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description
Release AHB2 peripherals reset.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD (*)
  - LL_AHB2_GRP1_PERIPH_GPIOE (*)
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG (*)
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_OTGFS (*)

- LL_AHB2_GRP1_PERIPH_ADC
- LL_AHB2_GRP1_PERIPH_DCMI (*)
- LL_AHB2_GRP1_PERIPH_AES (*)
- LL_AHB2_GRP1_PERIPH_HASH (*)
- LL_AHB2_GRP1_PERIPH_RNG
- LL_AHB2_GRP1_PERIPH_OSPIM (*)
- LL_AHB2_GRP1_PERIPH_SDMMC1 (*)

Return values	• **None:**
Reference Manual to LL API cross reference:	• AHB2RSTR GPIOARST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR GPIOBRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR GPIOCRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR GPIODRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR GPIOERST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR GPIOFRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR GPIOGRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR GPIOHRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR GPIOIRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR OTGFSRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR ADCRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR DCMIRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR AESRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR HASHRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR RNGRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR OSPIMRST LL_AHB2_GRP1_ReleaseReset • AHB2RSTR SDMMC1RST LL_AHB2_GRP1_ReleaseReset

LL_AHB2_GRP1_EnableClockStopSleep

Function name	**__STATIC_INLINE void LL_AHB2_GRP1_EnableClockStopSleep (uint32_t Periphs)**
Function description	Enable AHB2 peripheral clocks in Sleep and Stop modes.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB2_GRP1_PERIPH_GPIOA – LL_AHB2_GRP1_PERIPH_GPIOB – LL_AHB2_GRP1_PERIPH_GPIOC – LL_AHB2_GRP1_PERIPH_GPIOD (*) – LL_AHB2_GRP1_PERIPH_GPIOE (*) – LL_AHB2_GRP1_PERIPH_GPIOF (*) – LL_AHB2_GRP1_PERIPH_GPIOG (*) – LL_AHB2_GRP1_PERIPH_GPIOH – LL_AHB2_GRP1_PERIPH_GPIOI (*) – LL_AHB2_GRP1_PERIPH_SRAM2 – LL_AHB2_GRP1_PERIPH_SRAM3 (*) – LL_AHB2_GRP1_PERIPH_OTGFS (*) – LL_AHB2_GRP1_PERIPH_ADC – LL_AHB2_GRP1_PERIPH_DCMI (*) – LL_AHB2_GRP1_PERIPH_AES (*) – LL_AHB2_GRP1_PERIPH_HASH (*) – LL_AHB2_GRP1_PERIPH_RNG – LL_AHB2_GRP1_PERIPH_OSPIM (*)

– LL_AHB2_GRP1_PERIPH_SDMMC1 (*)

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2SMENR GPIOASMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOBSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOCSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIODSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOESMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOFSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOGSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOHSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOISMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR SRAM2SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR SRAM3SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR OTGFSSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR ADCSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR DCMISMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR AESSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR HASHSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR RNGSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR OSPIMSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR SDMMC1SMEN LL_AHB2_GRP1_EnableClockStopSleep

LL_AHB2_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable AHB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  – LL_AHB2_GRP1_PERIPH_GPIOA
  – LL_AHB2_GRP1_PERIPH_GPIOB
  – LL_AHB2_GRP1_PERIPH_GPIOC
  – LL_AHB2_GRP1_PERIPH_GPIOD (*)

- LL_AHB2_GRP1_PERIPH_GPIOE (*)
- LL_AHB2_GRP1_PERIPH_GPIOF (*)
- LL_AHB2_GRP1_PERIPH_GPIOG (*)
- LL_AHB2_GRP1_PERIPH_GPIOH
- LL_AHB2_GRP1_PERIPH_GPIOI (*)
- LL_AHB2_GRP1_PERIPH_SRAM2
- LL_AHB2_GRP1_PERIPH_SRAM3 (*)
- LL_AHB2_GRP1_PERIPH_OTGFS (*)
- LL_AHB2_GRP1_PERIPH_ADC
- LL_AHB2_GRP1_PERIPH_DCMI (*)
- LL_AHB2_GRP1_PERIPH_AES (*)
- LL_AHB2_GRP1_PERIPH_HASH (*)
- LL_AHB2_GRP1_PERIPH_RNG
- LL_AHB2_GRP1_PERIPH_OSPIM (*)
- LL_AHB2_GRP1_PERIPH_SDMMC1 (*)

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2SMENR GPIOASMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOBSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOCSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIODSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOESMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOFSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOGSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOHSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOISMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR SRAM2SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR SRAM3SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR OTGFSSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR ADCSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR DCMISMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR AESSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR HASHSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR RNGSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR OSPIMSMEN LL_AHB2_GRP1_DisableClockStopSleep

- AHB2SMENR SDMMC1SMEN LL_AHB2_GRP1_DisableClockStopSleep

LL_AHB3_GRP1_EnableClock

Function name	**__STATIC_INLINE void LL_AHB3_GRP1_EnableClock (uint32_t Periphs)**
Function description	Enable AHB3 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB3_GRP1_PERIPH_FMC (*) – LL_AHB3_GRP1_PERIPH_QSPI (*) – LL_AHB3_GRP1_PERIPH_OSPI1 (*) – LL_AHB3_GRP1_PERIPH_OSPI2 (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• AHB3ENR FMCEN LL_AHB3_GRP1_EnableClock • AHB3ENR QSPIEN LL_AHB3_GRP1_EnableClock • AHB3ENR OSPI1EN LL_AHB3_GRP1_EnableClock • AHB3ENR OSPI2EN LL_AHB3_GRP1_EnableClock

LL_AHB3_GRP1_IsEnabledClock

Function name	**__STATIC_INLINE uint32_t LL_AHB3_GRP1_IsEnabledClock (uint32_t Periphs)**
Function description	Check if AHB3 peripheral clock is enabled or not.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB3_GRP1_PERIPH_FMC (*) – LL_AHB3_GRP1_PERIPH_QSPI (*) – LL_AHB3_GRP1_PERIPH_OSPI1 (*) – LL_AHB3_GRP1_PERIPH_OSPI2 (*)
Return values	• **State:** of Periphs (1 or 0).
Reference Manual to LL API cross reference:	• AHB3ENR FMCEN LL_AHB3_GRP1_IsEnabledClock • AHB3ENR QSPIEN LL_AHB3_GRP1_IsEnabledClock • AHB3ENR OSPI1EN LL_AHB3_GRP1_IsEnabledClock • AHB3ENR OSPI2EN LL_AHB3_GRP1_IsEnabledClock

LL_AHB3_GRP1_DisableClock

Function name	**__STATIC_INLINE void LL_AHB3_GRP1_DisableClock (uint32_t Periphs)**
Function description	Disable AHB3 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB3_GRP1_PERIPH_FMC (*) – LL_AHB3_GRP1_PERIPH_QSPI (*) – LL_AHB3_GRP1_PERIPH_OSPI1 (*) – LL_AHB3_GRP1_PERIPH_OSPI2 (*)

Return values	• **None:**
Reference Manual to LL API cross reference:	• AHB3ENR FMCEN LL_AHB3_GRP1_DisableClock • AHB3ENR QSPIEN LL_AHB3_GRP1_DisableClock • AHB3ENR OSPI1EN LL_AHB3_GRP1_DisableClock • AHB3ENR OSPI2EN LL_AHB3_GRP1_DisableClock

LL_AHB3_GRP1_ForceReset

Function name	**__STATIC_INLINE void LL_AHB3_GRP1_ForceReset (uint32_t Periphs)**
Function description	Force AHB3 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB3_GRP1_PERIPH_ALL – LL_AHB3_GRP1_PERIPH_FMC (*) – LL_AHB3_GRP1_PERIPH_QSPI (*) – LL_AHB3_GRP1_PERIPH_OSPI1 (*) – LL_AHB3_GRP1_PERIPH_OSPI2 (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• AHB3RSTR FMCRST LL_AHB3_GRP1_ForceReset • AHB3RSTR QSPIRST LL_AHB3_GRP1_ForceReset • AHB3RSTR OSPI1RST LL_AHB3_GRP1_ForceReset • AHB3RSTR OSPI2RST LL_AHB3_GRP1_ForceReset

LL_AHB3_GRP1_ReleaseReset

Function name	**__STATIC_INLINE void LL_AHB3_GRP1_ReleaseReset (uint32_t Periphs)**
Function description	Release AHB3 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB2_GRP1_PERIPH_ALL – LL_AHB3_GRP1_PERIPH_FMC (*) – LL_AHB3_GRP1_PERIPH_QSPI (*) – LL_AHB3_GRP1_PERIPH_OSPI1 (*) – LL_AHB3_GRP1_PERIPH_OSPI2 (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• AHB3RSTR FMCRST LL_AHB3_GRP1_ReleaseReset • AHB3RSTR QSPIRST LL_AHB3_GRP1_ReleaseReset • AHB3RSTR OSPI1RST LL_AHB3_GRP1_ReleaseReset • AHB3RSTR OSPI2RST LL_AHB3_GRP1_ReleaseReset

LL_AHB3_GRP1_EnableClockStopSleep

Function name	**__STATIC_INLINE void LL_AHB3_GRP1_EnableClockStopSleep (uint32_t Periphs)**
Function description	Enable AHB3 peripheral clocks in Sleep and Stop modes.

Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB3_GRP1_PERIPH_FMC (*) – LL_AHB3_GRP1_PERIPH_QSPI (*) – LL_AHB3_GRP1_PERIPH_OSPI1 (*) – LL_AHB3_GRP1_PERIPH_OSPI2 (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• AHB3SMENR FMCSMEN LL_AHB3_GRP1_EnableClockStopSleep • AHB3SMENR QSPISMEN LL_AHB3_GRP1_EnableClockStopSleep • AHB3SMENR OSPI1SMEN LL_AHB3_GRP1_EnableClockStopSleep • AHB3SMENR OSPI2SMEN LL_AHB3_GRP1_EnableClockStopSleep

LL_AHB3_GRP1_DisableClockStopSleep

Function name	**__STATIC_INLINE void LL_AHB3_GRP1_DisableClockStopSleep (uint32_t Periphs)**
Function description	Disable AHB3 peripheral clocks in Sleep and Stop modes.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_AHB3_GRP1_PERIPH_FMC (*) – LL_AHB3_GRP1_PERIPH_QSPI (*) – LL_AHB3_GRP1_PERIPH_OSPI1 (*) – LL_AHB3_GRP1_PERIPH_OSPI2 (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• AHB3SMENR FMCSMEN LL_AHB3_GRP1_DisableClockStopSleep • AHB3SMENR QSPISMEN LL_AHB3_GRP1_DisableClockStopSleep • AHB3SMENR OSPI1SMEN LL_AHB3_GRP1_DisableClockStopSleep • AHB3SMENR OSPI2SMEN LL_AHB3_GRP1_DisableClockStopSleep •

LL_APB1_GRP1_EnableClock

Function name	**__STATIC_INLINE void LL_APB1_GRP1_EnableClock (uint32_t Periphs)**
Function description	Enable APB1 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM4 (*) – LL_APB1_GRP1_PERIPH_TIM5 (*)

- LL_APB1_GRP1_PERIPH_TIM6
- LL_APB1_GRP1_PERIPH_TIM7
- LL_APB1_GRP1_PERIPH_LCD (*)
- LL_APB1_GRP1_PERIPH_RTCAPB (*)
- LL_APB1_GRP1_PERIPH_WWDG
- LL_APB1_GRP1_PERIPH_SPI2 (*)
- LL_APB1_GRP1_PERIPH_SPI3
- LL_APB1_GRP1_PERIPH_USART2
- LL_APB1_GRP1_PERIPH_USART3 (*)
- LL_APB1_GRP1_PERIPH_UART4 (*)
- LL_APB1_GRP1_PERIPH_UART5 (*)
- LL_APB1_GRP1_PERIPH_I2C1
- LL_APB1_GRP1_PERIPH_I2C2 (*)
- LL_APB1_GRP1_PERIPH_I2C3
- LL_APB1_GRP1_PERIPH_CRS (*)
- LL_APB1_GRP1_PERIPH_CAN1
- LL_APB1_GRP1_PERIPH_CAN2 (*)
- LL_APB1_GRP1_PERIPH_USB (*)
- LL_APB1_GRP1_PERIPH_PWR
- LL_APB1_GRP1_PERIPH_DAC1
- LL_APB1_GRP1_PERIPH_OPAMP
- LL_APB1_GRP1_PERIPH_LPTIM1

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR1 TIM2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM3EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM4EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM5EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM6EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM7EN LL_APB1_GRP1_EnableClock
- APB1ENR1 LCDEN LL_APB1_GRP1_EnableClock
- APB1ENR1 RTCAPBEN LL_APB1_GRP1_EnableClock
- APB1ENR1 WWDGEN LL_APB1_GRP1_EnableClock
- APB1ENR1 SPI2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 SPI3EN LL_APB1_GRP1_EnableClock
- APB1ENR1 USART2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 USART3EN LL_APB1_GRP1_EnableClock
- APB1ENR1 UART4EN LL_APB1_GRP1_EnableClock
- APB1ENR1 UART5EN LL_APB1_GRP1_EnableClock
- APB1ENR1 I2C1EN LL_APB1_GRP1_EnableClock
- APB1ENR1 I2C2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 I2C3EN LL_APB1_GRP1_EnableClock
- APB1ENR1 CRSEN LL_APB1_GRP1_EnableClock
- APB1ENR1 CAN1EN LL_APB1_GRP1_EnableClock
- APB1ENR1 USBFSEN LL_APB1_GRP1_EnableClock
- APB1ENR1 CAN2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 PWREN LL_APB1_GRP1_EnableClock
- APB1ENR1 DAC1EN LL_APB1_GRP1_EnableClock
- APB1ENR1 OPAMPEN LL_APB1_GRP1_EnableClock
- APB1ENR1 LPTIM1EN LL_APB1_GRP1_EnableClock

LL_APB1_GRP2_EnableClock

Function name	**__STATIC_INLINE void LL_APB1_GRP2_EnableClock (uint32_t Periphs)**
Function description	Enable APB1 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP2_PERIPH_LPUART1 – LL_APB1_GRP2_PERIPH_I2C4 (*) – LL_APB1_GRP2_PERIPH_SWPMI1 (*) – LL_APB1_GRP2_PERIPH_LPTIM2
Return values	• **None:**
Reference Manual to LL API cross reference:	• APB1ENR2 LPUART1EN LL_APB1_GRP2_EnableClock • APB1ENR2 I2C4EN LL_APB1_GRP2_EnableClock • APB1ENR2 SWPMI1EN LL_APB1_GRP2_EnableClock • APB1ENR2 LPTIM2EN LL_APB1_GRP2_EnableClock

LL_APB1_GRP1_IsEnabledClock

Function name	**__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClock (uint32_t Periphs)**
Function description	Check if APB1 peripheral clock is enabled or not.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM4 (*) – LL_APB1_GRP1_PERIPH_TIM5 (*) – LL_APB1_GRP1_PERIPH_TIM6 – LL_APB1_GRP1_PERIPH_TIM7 – LL_APB1_GRP1_PERIPH_LCD (*) – LL_APB1_GRP1_PERIPH_RTCAPB (*) – LL_APB1_GRP1_PERIPH_WWDG – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_SPI3 – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_USART3 (*) – LL_APB1_GRP1_PERIPH_UART4 (*) – LL_APB1_GRP1_PERIPH_UART5 (*) – LL_APB1_GRP1_PERIPH_I2C1 – LL_APB1_GRP1_PERIPH_I2C2 (*) – LL_APB1_GRP1_PERIPH_I2C3 – LL_APB1_GRP1_PERIPH_CRS (*) – LL_APB1_GRP1_PERIPH_CAN1 – LL_APB1_GRP1_PERIPH_CAN2 (*) – LL_APB1_GRP1_PERIPH_USB (*) – LL_APB1_GRP1_PERIPH_PWR – LL_APB1_GRP1_PERIPH_DAC1 – LL_APB1_GRP1_PERIPH_OPAMP – LL_APB1_GRP1_PERIPH_LPTIM1

Return values	• **State:** of Periphs (1 or 0).
Reference Manual to LL API cross reference:	• APB1ENR1 TIM2EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 TIM3EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 TIM4EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 TIM5EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 TIM6EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 TIM7EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 LCDEN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 RTCAPBEN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 WWDGEN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 SPI2EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 SPI3EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 USART2EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 USART3EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 UART4EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 UART5EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 I2C1EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 I2C2EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 I2C3EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 CRSEN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 CAN1EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 USBFSEN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 CAN2EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 PWREN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 DAC1EN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 OPAMPEN LL_APB1_GRP1_IsEnabledClock • APB1ENR1 LPTIM1EN LL_APB1_GRP1_IsEnabledClock

LL_APB1_GRP2_IsEnabledClock

Function name	**__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClock (uint32_t Periphs)**
Function description	Check if APB1 peripheral clock is enabled or not.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP2_PERIPH_LPUART1 – LL_APB1_GRP2_PERIPH_I2C4 (*) – LL_APB1_GRP2_PERIPH_SWPMI1 (*) – LL_APB1_GRP2_PERIPH_LPTIM2
Return values	• **State:** of Periphs (1 or 0).
Reference Manual to LL API cross reference:	• APB1ENR2 LPUART1EN LL_APB1_GRP2_IsEnabledClock • APB1ENR2 I2C4EN LL_APB1_GRP2_IsEnabledClock • APB1ENR2 SWPMI1EN LL_APB1_GRP2_IsEnabledClock • APB1ENR2 LPTIM2EN LL_APB1_GRP2_IsEnabledClock

LL_APB1_GRP1_DisableClock

Function name	**__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periphs)**

Function description	Disable APB1 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM4 (*) – LL_APB1_GRP1_PERIPH_TIM5 (*) – LL_APB1_GRP1_PERIPH_TIM6 – LL_APB1_GRP1_PERIPH_TIM7 – LL_APB1_GRP1_PERIPH_LCD (*) – LL_APB1_GRP1_PERIPH_RTCAPB (*) – LL_APB1_GRP1_PERIPH_WWDG – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_SPI3 – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_USART3 (*) – LL_APB1_GRP1_PERIPH_UART4 (*) – LL_APB1_GRP1_PERIPH_UART5 (*) – LL_APB1_GRP1_PERIPH_I2C1 – LL_APB1_GRP1_PERIPH_I2C2 (*) – LL_APB1_GRP1_PERIPH_I2C3 – LL_APB1_GRP1_PERIPH_CRS (*) – LL_APB1_GRP1_PERIPH_CAN1 – LL_APB1_GRP1_PERIPH_CAN2 (*) – LL_APB1_GRP1_PERIPH_USB (*) – LL_APB1_GRP1_PERIPH_PWR – LL_APB1_GRP1_PERIPH_DAC1 – LL_APB1_GRP1_PERIPH_OPAMP – LL_APB1_GRP1_PERIPH_LPTIM1
Return values	• **None:**
Reference Manual to LL API cross reference:	• APB1ENR1 TIM2EN LL_APB1_GRP1_DisableClock • APB1ENR1 TIM3EN LL_APB1_GRP1_DisableClock • APB1ENR1 TIM4EN LL_APB1_GRP1_DisableClock • APB1ENR1 TIM5EN LL_APB1_GRP1_DisableClock • APB1ENR1 TIM6EN LL_APB1_GRP1_DisableClock • APB1ENR1 TIM7EN LL_APB1_GRP1_DisableClock • APB1ENR1 LCDEN LL_APB1_GRP1_DisableClock • APB1ENR1 RTCAPBEN LL_APB1_GRP1_DisableClock • APB1ENR1 WWDGEN LL_APB1_GRP1_DisableClock • APB1ENR1 SPI2EN LL_APB1_GRP1_DisableClock • APB1ENR1 SPI3EN LL_APB1_GRP1_DisableClock • APB1ENR1 USART2EN LL_APB1_GRP1_DisableClock • APB1ENR1 USART3EN LL_APB1_GRP1_DisableClock • APB1ENR1 UART4EN LL_APB1_GRP1_DisableClock • APB1ENR1 UART5EN LL_APB1_GRP1_DisableClock • APB1ENR1 I2C1EN LL_APB1_GRP1_DisableClock • APB1ENR1 I2C2EN LL_APB1_GRP1_DisableClock • APB1ENR1 I2C3EN LL_APB1_GRP1_DisableClock • APB1ENR1 CRSEN LL_APB1_GRP1_DisableClock • APB1ENR1 CAN1EN LL_APB1_GRP1_DisableClock • APB1ENR1 USBFSEN LL_APB1_GRP1_DisableClock

- APB1ENR1 CAN2EN LL_APB1_GRP1_DisableClock
- APB1ENR1 PWREN LL_APB1_GRP1_DisableClock
- APB1ENR1 DAC1EN LL_APB1_GRP1_DisableClock
- APB1ENR1 OPAMPEN LL_APB1_GRP1_DisableClock
- APB1ENR1 LPTIM1EN LL_APB1_GRP1_DisableClock

LL_APB1_GRP2_DisableClock

Function name	**__STATIC_INLINE void LL_APB1_GRP2_DisableClock (uint32_t Periphs)**
Function description	Disable APB1 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP2_PERIPH_LPUART1 – LL_APB1_GRP2_PERIPH_I2C4 (*) – LL_APB1_GRP2_PERIPH_SWPMI1 (*) – LL_APB1_GRP2_PERIPH_LPTIM2
Return values	• **None:**
Reference Manual to LL API cross reference:	• APB1ENR2 LPUART1EN LL_APB1_GRP2_DisableClock • APB1ENR2 I2C4EN LL_APB1_GRP2_DisableClock • APB1ENR2 SWPMI1EN LL_APB1_GRP2_DisableClock • APB1ENR2 LPTIM2EN LL_APB1_GRP2_DisableClock

LL_APB1_GRP1_ForceReset

Function name	**__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)**
Function description	Force APB1 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_ALL – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM4 (*) – LL_APB1_GRP1_PERIPH_TIM5 (*) – LL_APB1_GRP1_PERIPH_TIM6 – LL_APB1_GRP1_PERIPH_TIM7 – LL_APB1_GRP1_PERIPH_LCD (*) – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_SPI3 – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_USART3 (*) – LL_APB1_GRP1_PERIPH_UART4 (*) – LL_APB1_GRP1_PERIPH_UART5 (*) – LL_APB1_GRP1_PERIPH_I2C1 – LL_APB1_GRP1_PERIPH_I2C2 (*) – LL_APB1_GRP1_PERIPH_I2C3 – LL_APB1_GRP1_PERIPH_CRS (*) – LL_APB1_GRP1_PERIPH_CAN1 – LL_APB1_GRP1_PERIPH_CAN2 (*)

- LL_APB1_GRP1_PERIPH_USB (*)
- LL_APB1_GRP1_PERIPH_PWR
- LL_APB1_GRP1_PERIPH_DAC1
- LL_APB1_GRP1_PERIPH_OPAMP
- LL_APB1_GRP1_PERIPH_LPTIM1

Return values
- **None:**

Reference Manual to LL API cross reference:
- APB1RSTR1 TIM2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM3RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM4RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM5RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM6RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM7RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 LCDRST LL_APB1_GRP1_ForceReset
- APB1RSTR1 SPI2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 SPI3RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 USART2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 USART3RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 UART4RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 UART5RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 I2C1RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 I2C2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 I2C3RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 CRSRST LL_APB1_GRP1_ForceReset
- APB1RSTR1 CAN1RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 USBFSRST LL_APB1_GRP1_ForceReset
- APB1RSTR1 CAN2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 PWRRST LL_APB1_GRP1_ForceReset
- APB1RSTR1 DAC1RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 OPAMPRST LL_APB1_GRP1_ForceReset
- APB1RSTR1 LPTIM1RST LL_APB1_GRP1_ForceReset

LL_APB1_GRP2_ForceReset

Function name
__STATIC_INLINE void LL_APB1_GRP2_ForceReset (uint32_t Periphs)

Function description
Force APB1 peripherals reset.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_LPUART1
  - LL_APB1_GRP2_PERIPH_I2C4 (*)
  - LL_APB1_GRP2_PERIPH_SWPMI1 (*)
  - LL_APB1_GRP2_PERIPH_LPTIM2

Return values
- **None:**

Reference Manual to LL API cross reference:
- APB1RSTR2 LPUART1RST LL_APB1_GRP2_ForceReset
- APB1RSTR2 I2C4RST LL_APB1_GRP2_ForceReset
- APB1RSTR2 SWPMI1RST LL_APB1_GRP2_ForceReset
- APB1RSTR2 LPTIM2RST LL_APB1_GRP2_ForceReset

LL_APB1_GRP1_ReleaseReset

Function name	**__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periphs)**
Function description	Release APB1 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_ALL – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM4 (*) – LL_APB1_GRP1_PERIPH_TIM5 (*) – LL_APB1_GRP1_PERIPH_TIM6 – LL_APB1_GRP1_PERIPH_TIM7 – LL_APB1_GRP1_PERIPH_LCD (*) – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_SPI3 – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_USART3 (*) – LL_APB1_GRP1_PERIPH_UART4 (*) – LL_APB1_GRP1_PERIPH_UART5 (*) – LL_APB1_GRP1_PERIPH_I2C1 – LL_APB1_GRP1_PERIPH_I2C2 (*) – LL_APB1_GRP1_PERIPH_I2C3 – LL_APB1_GRP1_PERIPH_CRS (*) – LL_APB1_GRP1_PERIPH_CAN1 – LL_APB1_GRP1_PERIPH_CAN2 (*) – LL_APB1_GRP1_PERIPH_USB (*) – LL_APB1_GRP1_PERIPH_PWR – LL_APB1_GRP1_PERIPH_DAC1 – LL_APB1_GRP1_PERIPH_OPAMP – LL_APB1_GRP1_PERIPH_LPTIM1
Return values	• **None:**
Reference Manual to LL API cross reference:	• APB1RSTR1 TIM2RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 TIM3RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 TIM4RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 TIM5RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 TIM6RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 TIM7RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 LCDRST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 SPI2RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 SPI3RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 USART2RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 USART3RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 UART4RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 UART5RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 I2C1RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 I2C2RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 I2C3RST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 CRSRST LL_APB1_GRP1_ReleaseReset • APB1RSTR1 CAN1RST LL_APB1_GRP1_ReleaseReset

- APB1RSTR1 USBFSRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 CAN2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 PWRRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 DAC1RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 OPAMPRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 LPTIM1RST LL_APB1_GRP1_ReleaseReset

LL_APB1_GRP2_ReleaseReset

Function name
: **__STATIC_INLINE void LL_APB1_GRP2_ReleaseReset (uint32_t Periphs)**

Function description
: Release APB1 peripherals reset.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_LPUART1
  - LL_APB1_GRP2_PERIPH_I2C4 (*)
  - LL_APB1_GRP2_PERIPH_SWPMI1 (*)
  - LL_APB1_GRP2_PERIPH_LPTIM2

Return values
- **None:**

Reference Manual to LL API cross reference:
- APB1RSTR2 LPUART1RST LL_APB1_GRP2_ReleaseReset
- APB1RSTR2 I2C4RST LL_APB1_GRP2_ReleaseReset
- APB1RSTR2 SWPMI1RST LL_APB1_GRP2_ReleaseReset
- APB1RSTR2 LPTIM2RST LL_APB1_GRP2_ReleaseReset

LL_APB1_GRP1_EnableClockStopSleep

Function name
: **__STATIC_INLINE void LL_APB1_GRP1_EnableClockStopSleep (uint32_t Periphs)**

Function description
: Enable APB1 peripheral clocks in Sleep and Stop modes.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3 (*)
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_LCD (*)
  - LL_APB1_GRP1_PERIPH_RTCAPB (*)
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_I2C3

- LL_APB1_GRP1_PERIPH_CRS (*)
- LL_APB1_GRP1_PERIPH_CAN1
- LL_APB1_GRP1_PERIPH_CAN2 (*)
- LL_APB1_GRP1_PERIPH_USB (*)
- LL_APB1_GRP1_PERIPH_PWR
- LL_APB1_GRP1_PERIPH_DAC1
- LL_APB1_GRP1_PERIPH_OPAMP
- LL_APB1_GRP1_PERIPH_LPTIM1

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1SMENR1 TIM2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM4SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM5SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM6SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM7SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 LCDSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 RTCAPBSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 WWDGSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 SPI2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 SPI3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 USART2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 USART3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 UART4SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 UART5SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 I2C1SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 I2C2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 I2C3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 CRSSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 CAN1SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 USBFSSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 CAN2SMEN LL_APB1_GRP1_EnableClockStopSleep

- APB1SMENR1 PWRSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 DAC1SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 OPAMPSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 LPTIM1SMEN LL_APB1_GRP1_EnableClockStopSleep

LL_APB1_GRP2_EnableClockStopSleep

Function name	**__STATIC_INLINE void LL_APB1_GRP2_EnableClockStopSleep (uint32_t Periphs)**
Function description	Enable APB1 peripheral clocks in Sleep and Stop modes.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP2_PERIPH_LPUART1 – LL_APB1_GRP2_PERIPH_I2C4 (*) – LL_APB1_GRP2_PERIPH_SWPMI1 (*) – LL_APB1_GRP2_PERIPH_LPTIM2
Return values	• **None:**
Reference Manual to LL API cross reference:	• APB1SMENR2 LPUART1SMEN LL_APB1_GRP2_EnableClockStopSleep • APB1SMENR2 I2C4SMEN LL_APB1_GRP2_EnableClockStopSleep • APB1SMENR2 SWPMI1SMEN LL_APB1_GRP2_EnableClockStopSleep • APB1SMENR2 LPTIM2SMEN LL_APB1_GRP2_EnableClockStopSleep

LL_APB1_GRP1_DisableClockStopSleep

Function name	**__STATIC_INLINE void LL_APB1_GRP1_DisableClockStopSleep (uint32_t Periphs)**
Function description	Disable APB1 peripheral clocks in Sleep and Stop modes.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP1_PERIPH_TIM2 – LL_APB1_GRP1_PERIPH_TIM3 (*) – LL_APB1_GRP1_PERIPH_TIM4 (*) – LL_APB1_GRP1_PERIPH_TIM5 (*) – LL_APB1_GRP1_PERIPH_TIM6 – LL_APB1_GRP1_PERIPH_TIM7 – LL_APB1_GRP1_PERIPH_LCD (*) – LL_APB1_GRP1_PERIPH_RTCAPB (*) – LL_APB1_GRP1_PERIPH_WWDG – LL_APB1_GRP1_PERIPH_SPI2 (*) – LL_APB1_GRP1_PERIPH_SPI3 – LL_APB1_GRP1_PERIPH_USART2 – LL_APB1_GRP1_PERIPH_USART3 (*)

- LL_APB1_GRP1_PERIPH_UART4 (*)
- LL_APB1_GRP1_PERIPH_UART5 (*)
- LL_APB1_GRP1_PERIPH_I2C1
- LL_APB1_GRP1_PERIPH_I2C2 (*)
- LL_APB1_GRP1_PERIPH_I2C3
- LL_APB1_GRP1_PERIPH_CRS (*)
- LL_APB1_GRP1_PERIPH_CAN1
- LL_APB1_GRP1_PERIPH_CAN2 (*)
- LL_APB1_GRP1_PERIPH_USB (*)
- LL_APB1_GRP1_PERIPH_PWR
- LL_APB1_GRP1_PERIPH_DAC1
- LL_APB1_GRP1_PERIPH_OPAMP
- LL_APB1_GRP1_PERIPH_LPTIM1

Return values
- **None:**

Reference Manual to LL API cross reference:
- APB1SMENR1 TIM2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM4SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM5SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM6SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM7SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 LCDSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 RTCAPBSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 WWDGSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 SPI2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 SPI3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 USART2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 USART3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 UART4SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 UART5SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 I2C1SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 I2C2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 I2C3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 CRSSMEN LL_APB1_GRP1_DisableClockStopSleep

- APB1SMENR1 CAN1SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 USBFSSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 CAN2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 PWRSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 DAC1SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 OPAMPSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 LPTIM1SMEN LL_APB1_GRP1_DisableClockStopSleep

LL_APB1_GRP2_DisableClockStopSleep

Function name	**__STATIC_INLINE void LL_APB1_GRP2_DisableClockStopSleep (uint32_t Periphs)**
Function description	Disable APB1 peripheral clocks in Sleep and Stop modes.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB1_GRP2_PERIPH_LPUART1 – LL_APB1_GRP2_PERIPH_I2C4 (*) – LL_APB1_GRP2_PERIPH_SWPMI1 (*) – LL_APB1_GRP2_PERIPH_LPTIM2
Return values	• **None:**
Reference Manual to LL API cross reference:	• APB1SMENR2 LPUART1SMEN LL_APB1_GRP2_DisableClockStopSleep • APB1SMENR2 I2C4SMEN LL_APB1_GRP2_DisableClockStopSleep • APB1SMENR2 SWPMI1SMEN LL_APB1_GRP2_DisableClockStopSleep • APB1SMENR2 LPTIM2SMEN LL_APB1_GRP2_DisableClockStopSleep

LL_APB2_GRP1_EnableClock

Function name	**__STATIC_INLINE void LL_APB2_GRP1_EnableClock (uint32_t Periphs)**
Function description	Enable APB2 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_FW – LL_APB2_GRP1_PERIPH_SDMMC1 (*) – LL_APB2_GRP1_PERIPH_TIM1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_TIM8 (*) – LL_APB2_GRP1_PERIPH_USART1

- LL_APB2_GRP1_PERIPH_TIM15
- LL_APB2_GRP1_PERIPH_TIM16
- LL_APB2_GRP1_PERIPH_TIM17 (*)
- LL_APB2_GRP1_PERIPH_SAI1
- LL_APB2_GRP1_PERIPH_SAI2 (*)
- LL_APB2_GRP1_PERIPH_DFSDM1 (*)
- LL_APB2_GRP1_PERIPH_LTDC (*)
- LL_APB2_GRP1_PERIPH_DSI (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- APB2ENR SYSCFGEN LL_APB2_GRP1_EnableClock
- APB2ENR FWEN LL_APB2_GRP1_EnableClock
- APB2ENR SDMMC1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM1EN LL_APB2_GRP1_EnableClock
- APB2ENR SPI1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM8EN LL_APB2_GRP1_EnableClock
- APB2ENR USART1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM15EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM16EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM17EN LL_APB2_GRP1_EnableClock
- APB2ENR SAI1EN LL_APB2_GRP1_EnableClock
- APB2ENR SAI2EN LL_APB2_GRP1_EnableClock
- APB2ENR DFSDM1EN LL_APB2_GRP1_EnableClock
- APB2ENR LTDCEN LL_APB2_GRP1_EnableClock
- APB2ENR DSIEN LL_APB2_GRP1_EnableClock

LL_APB2_GRP1_IsEnabledClock

Function name
__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description
Check if APB2 peripheral clock is enabled or not.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB2_GRP1_PERIPH_SYSCFG
  - LL_APB2_GRP1_PERIPH_FW
  - LL_APB2_GRP1_PERIPH_SDMMC1 (*)
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1 (*)
  - LL_APB2_GRP1_PERIPH_LTDC (*)
  - LL_APB2_GRP1_PERIPH_DSI (*)

Return values
- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross
- APB2ENR SYSCFGEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR FWEN LL_APB2_GRP1_IsEnabledClock

reference:	• APB2ENR SDMMC1EN LL_APB2_GRP1_IsEnabledClock • APB2ENR TIM1EN LL_APB2_GRP1_IsEnabledClock • APB2ENR SPI1EN LL_APB2_GRP1_IsEnabledClock • APB2ENR TIM8EN LL_APB2_GRP1_IsEnabledClock • APB2ENR USART1EN LL_APB2_GRP1_IsEnabledClock • APB2ENR TIM15EN LL_APB2_GRP1_IsEnabledClock • APB2ENR TIM16EN LL_APB2_GRP1_IsEnabledClock • APB2ENR TIM17EN LL_APB2_GRP1_IsEnabledClock • APB2ENR SAI1EN LL_APB2_GRP1_IsEnabledClock • APB2ENR SAI2EN LL_APB2_GRP1_IsEnabledClock • APB2ENR DFSDM1EN LL_APB2_GRP1_IsEnabledClock • APB2ENR LTDCEN LL_APB2_GRP1_IsEnabledClock • APB2ENR DSIEN LL_APB2_GRP1_IsEnabledClock

LL_APB2_GRP1_DisableClock

Function name	**__STATIC_INLINE void LL_APB2_GRP1_DisableClock (uint32_t Periphs)**
Function description	Disable APB2 peripherals clock.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_SDMMC1 (*) – LL_APB2_GRP1_PERIPH_TIM1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_TIM8 (*) – LL_APB2_GRP1_PERIPH_USART1 – LL_APB2_GRP1_PERIPH_TIM15 – LL_APB2_GRP1_PERIPH_TIM16 – LL_APB2_GRP1_PERIPH_TIM17 (*) – LL_APB2_GRP1_PERIPH_SAI1 – LL_APB2_GRP1_PERIPH_SAI2 (*) – LL_APB2_GRP1_PERIPH_DFSDM1 (*) – LL_APB2_GRP1_PERIPH_LTDC (*) – LL_APB2_GRP1_PERIPH_DSI (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• APB2ENR SYSCFGEN LL_APB2_GRP1_DisableClock • APB2ENR SDMMC1EN LL_APB2_GRP1_DisableClock • APB2ENR TIM1EN LL_APB2_GRP1_DisableClock • APB2ENR SPI1EN LL_APB2_GRP1_DisableClock • APB2ENR TIM8EN LL_APB2_GRP1_DisableClock • APB2ENR USART1EN LL_APB2_GRP1_DisableClock • APB2ENR TIM15EN LL_APB2_GRP1_DisableClock • APB2ENR TIM16EN LL_APB2_GRP1_DisableClock • APB2ENR TIM17EN LL_APB2_GRP1_DisableClock • APB2ENR SAI1EN LL_APB2_GRP1_DisableClock • APB2ENR SAI2EN LL_APB2_GRP1_DisableClock • APB2ENR DFSDM1EN LL_APB2_GRP1_DisableClock • APB2ENR LTDCEN LL_APB2_GRP1_DisableClock • APB2ENR DSIEN LL_APB2_GRP1_DisableClock

LL_APB2_GRP1_ForceReset

Function name	**__STATIC_INLINE void LL_APB2_GRP1_ForceReset (uint32_t Periphs)**
Function description	Force APB2 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_ALL – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_SDMMC1 (*) – LL_APB2_GRP1_PERIPH_TIM1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_TIM8 (*) – LL_APB2_GRP1_PERIPH_USART1 – LL_APB2_GRP1_PERIPH_TIM15 – LL_APB2_GRP1_PERIPH_TIM16 – LL_APB2_GRP1_PERIPH_TIM17 (*) – LL_APB2_GRP1_PERIPH_SAI1 – LL_APB2_GRP1_PERIPH_SAI2 (*) – LL_APB2_GRP1_PERIPH_DFSDM1 (*) – LL_APB2_GRP1_PERIPH_LTDC (*) – LL_APB2_GRP1_PERIPH_DSI (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• APB2RSTR SYSCFGRST LL_APB2_GRP1_ForceReset • APB2RSTR SDMMC1RST LL_APB2_GRP1_ForceReset • APB2RSTR TIM1RST LL_APB2_GRP1_ForceReset • APB2RSTR SPI1RST LL_APB2_GRP1_ForceReset • APB2RSTR TIM8RST LL_APB2_GRP1_ForceReset • APB2RSTR USART1RST LL_APB2_GRP1_ForceReset • APB2RSTR TIM15RST LL_APB2_GRP1_ForceReset • APB2RSTR TIM16RST LL_APB2_GRP1_ForceReset • APB2RSTR TIM17RST LL_APB2_GRP1_ForceReset • APB2RSTR SAI1RST LL_APB2_GRP1_ForceReset • APB2RSTR SAI2RST LL_APB2_GRP1_ForceReset • APB2RSTR DFSDM1RST LL_APB2_GRP1_ForceReset • APB2RSTR LTDCRST LL_APB2_GRP1_ForceReset • APB2RSTR DSIRST LL_APB2_GRP1_ForceReset

LL_APB2_GRP1_ReleaseReset

Function name	**__STATIC_INLINE void LL_APB2_GRP1_ReleaseReset (uint32_t Periphs)**
Function description	Release APB2 peripherals reset.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_ALL – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_SDMMC1 (*) – LL_APB2_GRP1_PERIPH_TIM1 – LL_APB2_GRP1_PERIPH_SPI1

- LL_APB2_GRP1_PERIPH_TIM8 (*)
- LL_APB2_GRP1_PERIPH_USART1
- LL_APB2_GRP1_PERIPH_TIM15
- LL_APB2_GRP1_PERIPH_TIM16
- LL_APB2_GRP1_PERIPH_TIM17 (*)
- LL_APB2_GRP1_PERIPH_SAI1
- LL_APB2_GRP1_PERIPH_SAI2 (*)
- LL_APB2_GRP1_PERIPH_DFSDM1 (*)
- LL_APB2_GRP1_PERIPH_LTDC (*)
- LL_APB2_GRP1_PERIPH_DSI (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- APB2RSTR SYSCFGRST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SDMMC1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SPI1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR USART1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SAI1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SAI2RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR DFSDM1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR LTDCRST LL_APB2_GRP1_ReleaseReset
- APB2RSTR DSIRST LL_APB2_GRP1_ReleaseReset

LL_APB2_GRP1_EnableClockStopSleep

Function name
__STATIC_INLINE void LL_APB2_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description
Enable APB2 peripheral clocks in Sleep and Stop modes.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB2_GRP1_PERIPH_SYSCFG
  - LL_APB2_GRP1_PERIPH_SDMMC1 (*)
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1 (*)
  - LL_APB2_GRP1_PERIPH_LTDC (*)
  - LL_APB2_GRP1_PERIPH_DSI (*)

Return values
- **None:**

Reference Manual to LL API cross
- APB2SMENR SYSCFGSMEN LL_APB2_GRP1_EnableClockStopSleep

reference:	• APB2SMENR SDMMC1SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR TIM1SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR SPI1SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR TIM8SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR USART1SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR TIM15SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR TIM16SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR TIM17SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR SAI1SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR SAI2SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR DFSDM1SMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR LTDCSMEN LL_APB2_GRP1_EnableClockStopSleep • APB2SMENR DSISMEN LL_APB2_GRP1_EnableClockStopSleep

LL_APB2_GRP1_DisableClockStopSleep

Function name	**__STATIC_INLINE void LL_APB2_GRP1_DisableClockStopSleep (uint32_t Periphs)**
Function description	Disable APB2 peripheral clocks in Sleep and Stop modes.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_APB2_GRP1_PERIPH_SYSCFG – LL_APB2_GRP1_PERIPH_SDMMC1 (*) – LL_APB2_GRP1_PERIPH_TIM1 – LL_APB2_GRP1_PERIPH_SPI1 – LL_APB2_GRP1_PERIPH_TIM8 (*) – LL_APB2_GRP1_PERIPH_USART1 – LL_APB2_GRP1_PERIPH_TIM15 – LL_APB2_GRP1_PERIPH_TIM16 – LL_APB2_GRP1_PERIPH_TIM17 (*) – LL_APB2_GRP1_PERIPH_SAI1 – LL_APB2_GRP1_PERIPH_SAI2 (*) – LL_APB2_GRP1_PERIPH_DFSDM1 (*) – LL_APB2_GRP1_PERIPH_LTDC (*) – LL_APB2_GRP1_PERIPH_DSI (*)
Return values	• **None:**
Reference Manual to LL API cross	• APB2SMENR SYSCFGSMEN LL_APB2_GRP1_DisableClockStopSleep

reference:

- APB2SMENR SDMMC1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR SPI1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM8SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR USART1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM15SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM16SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM17SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR SAI1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR SAI2SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR DFSDM1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR LTDCSMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR DSISMEN LL_APB2_GRP1_DisableClockStopSleep

74.2 BUS Firmware driver defines

74.2.1 BUS

AHB1 GRP1 PERIPH

LL_AHB1_GRP1_PERIPH_ALL

LL_AHB1_GRP1_PERIPH_DMA1

LL_AHB1_GRP1_PERIPH_DMA2

LL_AHB1_GRP1_PERIPH_DMAMUX1

LL_AHB1_GRP1_PERIPH_FLASH

LL_AHB1_GRP1_PERIPH_CRC

LL_AHB1_GRP1_PERIPH_TSC

LL_AHB1_GRP1_PERIPH_DMA2D

LL_AHB1_GRP1_PERIPH_GFXMMU

LL_AHB1_GRP1_PERIPH_SRAM1

AHB2 GRP1 PERIPH

LL_AHB2_GRP1_PERIPH_ALL

LL_AHB2_GRP1_PERIPH_GPIOA

LL_AHB2_GRP1_PERIPH_GPIOB

LL_AHB2_GRP1_PERIPH_GPIOC

LL_AHB2_GRP1_PERIPH_GPIOD

LL_AHB2_GRP1_PERIPH_GPIOE

LL_AHB2_GRP1_PERIPH_GPIOF

LL_AHB2_GRP1_PERIPH_GPIOG

LL_AHB2_GRP1_PERIPH_GPIOH

LL_AHB2_GRP1_PERIPH_GPIOI

LL_AHB2_GRP1_PERIPH_OTGFS

LL_AHB2_GRP1_PERIPH_ADC

LL_AHB2_GRP1_PERIPH_DCMI

LL_AHB2_GRP1_PERIPH_AES

LL_AHB2_GRP1_PERIPH_HASH

LL_AHB2_GRP1_PERIPH_RNG

LL_AHB2_GRP1_PERIPH_OSPIM

LL_AHB2_GRP1_PERIPH_SDMMC1

LL_AHB2_GRP1_PERIPH_SRAM2

LL_AHB2_GRP1_PERIPH_SRAM3

AHB3 GRP1 PERIPH

LL_AHB3_GRP1_PERIPH_ALL

LL_AHB3_GRP1_PERIPH_FMC

LL_AHB3_GRP1_PERIPH_OSPI1

LL_AHB3_GRP1_PERIPH_OSPI2

APB1 GRP1 PERIPH

LL_APB1_GRP1_PERIPH_ALL

LL_APB1_GRP1_PERIPH_TIM2

LL_APB1_GRP1_PERIPH_TIM3

LL_APB1_GRP1_PERIPH_TIM4

LL_APB1_GRP1_PERIPH_TIM5

LL_APB1_GRP1_PERIPH_TIM6

LL_APB1_GRP1_PERIPH_TIM7

LL_APB1_GRP1_PERIPH_RTCAPB

LL_APB1_GRP1_PERIPH_WWDG

LL_APB1_GRP1_PERIPH_SPI2

LL_APB1_GRP1_PERIPH_SPI3

LL_APB1_GRP1_PERIPH_USART2

LL_APB1_GRP1_PERIPH_USART3

LL_APB1_GRP1_PERIPH_UART4

LL_APB1_GRP1_PERIPH_UART5

LL_APB1_GRP1_PERIPH_I2C1

LL_APB1_GRP1_PERIPH_I2C2

LL_APB1_GRP1_PERIPH_I2C3

LL_APB1_GRP1_PERIPH_CRS

LL_APB1_GRP1_PERIPH_CAN1

LL_APB1_GRP1_PERIPH_PWR

LL_APB1_GRP1_PERIPH_DAC1

LL_APB1_GRP1_PERIPH_OPAMP

LL_APB1_GRP1_PERIPH_LPTIM1

APB1 GRP2 PERIPH

LL_APB1_GRP2_PERIPH_ALL

LL_APB1_GRP2_PERIPH_LPUART1

LL_APB1_GRP2_PERIPH_I2C4

LL_APB1_GRP2_PERIPH_LPTIM2

APB2 GRP1 PERIPH

LL_APB2_GRP1_PERIPH_ALL

LL_APB2_GRP1_PERIPH_SYSCFG

LL_APB2_GRP1_PERIPH_FW

LL_APB2_GRP1_PERIPH_TIM1

LL_APB2_GRP1_PERIPH_SPI1

LL_APB2_GRP1_PERIPH_TIM8

LL_APB2_GRP1_PERIPH_USART1

LL_APB2_GRP1_PERIPH_TIM15

LL_APB2_GRP1_PERIPH_TIM16

LL_APB2_GRP1_PERIPH_TIM17

LL_APB2_GRP1_PERIPH_SAI1

LL_APB2_GRP1_PERIPH_SAI2

LL_APB2_GRP1_PERIPH_DFSDM1

LL_APB2_GRP1_PERIPH_LTDC

LL_APB2_GRP1_PERIPH_DSI

75 LL COMP Generic Driver

75.1 COMP Firmware driver registers structures

75.1.1 LL_COMP_InitTypeDef

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t InputPlus***
- ***uint32_t InputMinus***
- ***uint32_t InputHysteresis***
- ***uint32_t OutputPolarity***
- ***uint32_t OutputBlankingSource***

Field Documentation

- ***uint32_t LL_COMP_InitTypeDef::PowerMode***
  Set comparator operating mode to adjust power and speed. This parameter can be a value of ***COMP_LL_EC_POWERMODE***This feature can be modified afterwards using unitary function **LL_COMP_SetPowerMode()**.
- ***uint32_t LL_COMP_InitTypeDef::InputPlus***
  Set comparator input plus (non-inverting input). This parameter can be a value of ***COMP_LL_EC_INPUT_PLUS***This feature can be modified afterwards using unitary function **LL_COMP_SetInputPlus()**.
- ***uint32_t LL_COMP_InitTypeDef::InputMinus***
  Set comparator input minus (inverting input). This parameter can be a value of ***COMP_LL_EC_INPUT_MINUS***This feature can be modified afterwards using unitary function **LL_COMP_SetInputMinus()**.
- ***uint32_t LL_COMP_InitTypeDef::InputHysteresis***
  Set comparator hysteresis mode of the input minus. This parameter can be a value of ***COMP_LL_EC_INPUT_HYSTERESIS***This feature can be modified afterwards using unitary function **LL_COMP_SetInputHysteresis()**.
- ***uint32_t LL_COMP_InitTypeDef::OutputPolarity***
  Set comparator output polarity. This parameter can be a value of ***COMP_LL_EC_OUTPUT_POLARITY***This feature can be modified afterwards using unitary function **LL_COMP_SetOutputPolarity()**.
- ***uint32_t LL_COMP_InitTypeDef::OutputBlankingSource***
  Set comparator blanking source. This parameter can be a value of ***COMP_LL_EC_OUTPUT_BLANKING_SOURCE***This feature can be modified afterwards using unitary function **LL_COMP_SetOutputBlankingSource()**.

75.2 COMP Firmware driver API description

75.2.1 Detailed description of functions

LL_COMP_SetCommonWindowMode

Function name	**__STATIC_INLINE void LL_COMP_SetCommonWindowMode (COMP_Common_TypeDef * COMPxy_COMMON, uint32_t WindowMode)**
Function	Set window mode of a pair of comparators instances (2 consecutive COMP

description	instances odd and even COMP<x> and COMP<x+1>).
Parameters	• **COMPxy_COMMON:** Comparator common instance (can be set directly from CMSIS definition or by using helper macro __LL_COMP_COMMON_INSTANCE()) • **WindowMode:** This parameter can be one of the following values: – LL_COMP_WINDOWMODE_DISABLE – LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR WINMODE LL_COMP_SetCommonWindowMode

LL_COMP_GetCommonWindowMode

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetCommonWindowMode (COMP_Common_TypeDef * COMPxy_COMMON)**
Function description	Get window mode of a pair of comparators instances (2 consecutive COMP instances odd and even COMP<x> and COMP<x+1>).
Parameters	• **COMPxy_COMMON:** Comparator common instance (can be set directly from CMSIS definition or by using helper macro __LL_COMP_COMMON_INSTANCE())
Return values	• **Returned:** value can be one of the following values: – LL_COMP_WINDOWMODE_DISABLE – LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON
Reference Manual to LL API cross reference:	• CSR WINMODE LL_COMP_GetCommonWindowMode

LL_COMP_SetPowerMode

Function name	**__STATIC_INLINE void LL_COMP_SetPowerMode (COMP_TypeDef * COMPx, uint32_t PowerMode)**
Function description	Set comparator instance operating mode to adjust power and speed.
Parameters	• **COMPx:** Comparator instance • **PowerMode:** This parameter can be one of the following values: – LL_COMP_POWERMODE_HIGHSPEED – LL_COMP_POWERMODE_MEDIUMSPEED – LL_COMP_POWERMODE_ULTRALOWPOWER
Return values	• **None:**
Reference Manual to LL API cross	• CSR PWRMODE LL_COMP_SetPowerMode

reference:

LL_COMP_GetPowerMode

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetPowerMode (COMP_TypeDef * COMPx)**
Function description	Get comparator instance operating mode to adjust power and speed.
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: – LL_COMP_POWERMODE_HIGHSPEED – LL_COMP_POWERMODE_MEDIUMSPEED – LL_COMP_POWERMODE_ULTRALOWPOWER
Reference Manual to LL API cross reference:	• CSR PWRMODE LL_COMP_GetPowerMode

LL_COMP_ConfigInputs

Function name	**__STATIC_INLINE void LL_COMP_ConfigInputs (COMP_TypeDef * COMPx, uint32_t InputMinus, uint32_t InputPlus)**
Function description	Set comparator inputs minus (inverting) and plus (non-inverting).
Parameters	• **COMPx:** Comparator instance • **InputMinus:** This parameter can be one of the following values: (*) Parameter not available on all devices. – LL_COMP_INPUT_MINUS_1_4VREFINT – LL_COMP_INPUT_MINUS_1_2VREFINT – LL_COMP_INPUT_MINUS_3_4VREFINT – LL_COMP_INPUT_MINUS_VREFINT – LL_COMP_INPUT_MINUS_DAC1_CH1 – LL_COMP_INPUT_MINUS_DAC1_CH2 (*) – LL_COMP_INPUT_MINUS_IO1 – LL_COMP_INPUT_MINUS_IO2 – LL_COMP_INPUT_MINUS_IO3 (*) – LL_COMP_INPUT_MINUS_IO4 (*) – LL_COMP_INPUT_MINUS_IO5 (*) • **InputPlus:** This parameter can be one of the following values: (*) Parameter not available on all devices. – LL_COMP_INPUT_PLUS_IO1 – LL_COMP_INPUT_PLUS_IO2 – LL_COMP_INPUT_PLUS_IO3 (*)
Return values	• **None:**
Notes	• In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual. • On this STM32 serie, scaler bridge is configurable: to optimize power consumption, this function enables the voltage scaler bridge only when required (when selecting

	comparator input based on VrefInt: VrefInt or subdivision of VrefInt). For scaler bridge power consumption values, refer to device datasheet, parameter "IDDA(SCALER)".Voltage scaler requires a delay for voltage stabilization. Refer to device datasheet, parameter "tSTART_SCALER".Scaler bridge is common for all comparator instances, therefore if at least one of the comparator instance is requiring the scaler bridge, it remains enabled.
Reference Manual to LL API cross reference:	• CSR INMSEL LL_COMP_ConfigInputs • CSR INPSEL LL_COMP_ConfigInputs • CSR BRGEN LL_COMP_ConfigInputs • CSR SCALEN LL_COMP_ConfigInputs

LL_COMP_SetInputPlus

Function name	**__STATIC_INLINE void LL_COMP_SetInputPlus (COMP_TypeDef * COMPx, uint32_t InputPlus)**
Function description	Set comparator input plus (non-inverting).
Parameters	• **COMPx:** Comparator instance • **InputPlus:** This parameter can be one of the following values: (*) Parameter not available on all devices. – LL_COMP_INPUT_PLUS_IO1 – LL_COMP_INPUT_PLUS_IO2 – LL_COMP_INPUT_PLUS_IO3 (*)
Return values	• **None:**
Notes	• In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
Reference Manual to LL API cross reference:	• CSR INPSEL LL_COMP_SetInputPlus

LL_COMP_GetInputPlus

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetInputPlus (COMP_TypeDef * COMPx)**
Function description	Get comparator input plus (non-inverting).
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: (*) Parameter not available on all devices. – LL_COMP_INPUT_PLUS_IO1 – LL_COMP_INPUT_PLUS_IO2 – LL_COMP_INPUT_PLUS_IO3 (*)
Notes	• In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
Reference Manual to LL API cross	• CSR INPSEL LL_COMP_GetInputPlus

reference:

LL_COMP_SetInputMinus

Function name	**__STATIC_INLINE void LL_COMP_SetInputMinus (COMP_TypeDef * COMPx, uint32_t InputMinus)**
Function description	Set comparator input minus (inverting).
Parameters	• **COMPx:** Comparator instance • **InputMinus:** This parameter can be one of the following values: (*) Parameter not available on all devices. – LL_COMP_INPUT_MINUS_1_4VREFINT – LL_COMP_INPUT_MINUS_1_2VREFINT – LL_COMP_INPUT_MINUS_3_4VREFINT – LL_COMP_INPUT_MINUS_VREFINT – LL_COMP_INPUT_MINUS_DAC1_CH1 – LL_COMP_INPUT_MINUS_DAC1_CH2 (*) – LL_COMP_INPUT_MINUS_IO1 – LL_COMP_INPUT_MINUS_IO2 – LL_COMP_INPUT_MINUS_IO3 (*) – LL_COMP_INPUT_MINUS_IO4 (*) – LL_COMP_INPUT_MINUS_IO5 (*)
Return values	• **None:**
Notes	• In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual. • On this STM32 serie, scaler bridge is configurable: to optimize power consumption, this function enables the voltage scaler bridge only when required (when selecting comparator input based on VrefInt: VrefInt or subdivision of VrefInt). For scaler bridge power consumption values, refer to device datasheet, parameter "IDDA(SCALER)".Voltage scaler requires a delay for voltage stabilization. Refer to device datasheet, parameter "tSTART_SCALER".Scaler bridge is common for all comparator instances, therefore if at least one of the comparator instance is requiring the scaler bridge, it remains enabled.
Reference Manual to LL API cross reference:	• CSR INMSEL LL_COMP_SetInputMinus • CSR BRGEN LL_COMP_SetInputMinus • CSR SCALEN LL_COMP_SetInputMinus

LL_COMP_GetInputMinus

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetInputMinus (COMP_TypeDef * COMPx)**
Function description	Get comparator input minus (inverting).
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: (*) Parameter not available on all devices. – LL_COMP_INPUT_MINUS_1_4VREFINT – LL_COMP_INPUT_MINUS_1_2VREFINT

- – LL_COMP_INPUT_MINUS_3_4VREFINT
- – LL_COMP_INPUT_MINUS_VREFINT
- – LL_COMP_INPUT_MINUS_DAC1_CH1
- – LL_COMP_INPUT_MINUS_DAC1_CH2 (*)
- – LL_COMP_INPUT_MINUS_IO1
- – LL_COMP_INPUT_MINUS_IO2
- – LL_COMP_INPUT_MINUS_IO3 (*)
- – LL_COMP_INPUT_MINUS_IO4 (*)
- – LL_COMP_INPUT_MINUS_IO5 (*)

Notes
- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:
- CSR INMSEL LL_COMP_GetInputMinus
- CSR BRGEN LL_COMP_GetInputMinus
- CSR SCALEN LL_COMP_GetInputMinus

LL_COMP_SetInputHysteresis

Function name
__STATIC_INLINE void LL_COMP_SetInputHysteresis (COMP_TypeDef * COMPx, uint32_t InputHysteresis)

Function description
Set comparator instance hysteresis mode of the input minus (inverting input).

Parameters
- **COMPx:** Comparator instance
- **InputHysteresis:** This parameter can be one of the following values:
  - – LL_COMP_HYSTERESIS_NONE
  - – LL_COMP_HYSTERESIS_LOW
  - – LL_COMP_HYSTERESIS_MEDIUM
  - – LL_COMP_HYSTERESIS_HIGH

Return values
- **None:**

Reference Manual to LL API cross reference:
- CSR HYST LL_COMP_SetInputHysteresis

LL_COMP_GetInputHysteresis

Function name
__STATIC_INLINE uint32_t LL_COMP_GetInputHysteresis (COMP_TypeDef * COMPx)

Function description
Get comparator instance hysteresis mode of the minus (inverting) input.

Parameters
- **COMPx:** Comparator instance

Return values
- **Returned:** value can be one of the following values:
  - – LL_COMP_HYSTERESIS_NONE
  - – LL_COMP_HYSTERESIS_LOW
  - – LL_COMP_HYSTERESIS_MEDIUM
  - – LL_COMP_HYSTERESIS_HIGH

Reference Manual to LL API cross
- CSR HYST LL_COMP_GetInputHysteresis

reference:

LL_COMP_SetOutputPolarity

Function name	**__STATIC_INLINE void LL_COMP_SetOutputPolarity (COMP_TypeDef * COMPx, uint32_t OutputPolarity)**
Function description	Set comparator instance output polarity.
Parameters	• **COMPx:** Comparator instance • **OutputPolarity:** This parameter can be one of the following values: – LL_COMP_OUTPUTPOL_NONINVERTED – LL_COMP_OUTPUTPOL_INVERTED
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR POLARITY LL_COMP_SetOutputPolarity

LL_COMP_GetOutputPolarity

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetOutputPolarity (COMP_TypeDef * COMPx)**
Function description	Get comparator instance output polarity.
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: – LL_COMP_OUTPUTPOL_NONINVERTED – LL_COMP_OUTPUTPOL_INVERTED
Reference Manual to LL API cross reference:	• CSR POLARITY LL_COMP_GetOutputPolarity

LL_COMP_SetOutputBlankingSource

Function name	**__STATIC_INLINE void LL_COMP_SetOutputBlankingSource (COMP_TypeDef * COMPx, uint32_t BlankingSource)**
Function description	Set comparator instance blanking source.
Parameters	• **COMPx:** Comparator instance • **BlankingSource:** This parameter can be one of the following values: (1) Parameter availability depending on timer availability on the selected device. (2) On STM32L4, parameter available only on comparator instance: COMP1. (3) On STM32L4, parameter available only on comparator instance: COMP2. – LL_COMP_BLANKINGSRC_NONE – LL_COMP_BLANKINGSRC_TIM1_OC5_COMP1 (1)(2) – LL_COMP_BLANKINGSRC_TIM2_OC3_COMP1 (1)(2) – LL_COMP_BLANKINGSRC_TIM3_OC3_COMP1 (1)(2) – LL_COMP_BLANKINGSRC_TIM3_OC4_COMP2 (1)(3) – LL_COMP_BLANKINGSRC_TIM8_OC5_COMP2 (1)(3)

– LL_COMP_BLANKINGSRC_TIM15_OC1_COMP2 (1)(3)

Return values	• **None:**
Notes	• Blanking source may be specific to each comparator instance. Refer to description of parameters or to reference manual. • Availability of parameters of blanking source from timer depends on timers availability on the selected device.
Reference Manual to LL API cross reference:	• CSR BLANKING LL_COMP_SetOutputBlankingSource

LL_COMP_GetOutputBlankingSource

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetOutputBlankingSource (COMP_TypeDef * COMPx)**
Function description	Get comparator instance blanking source.
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: (1) Parameter availability depending on timer availability on the selected device. (2) On STM32L4, parameter available only on comparator instance: COMP1. (3) On STM32L4, parameter available only on comparator instance: COMP2. – LL_COMP_BLANKINGSRC_NONE – LL_COMP_BLANKINGSRC_TIM1_OC5_COMP1 (1)(2) – LL_COMP_BLANKINGSRC_TIM2_OC3_COMP1 (1)(2) – LL_COMP_BLANKINGSRC_TIM3_OC3_COMP1 (1)(2) – LL_COMP_BLANKINGSRC_TIM3_OC4_COMP2 (1)(3) – LL_COMP_BLANKINGSRC_TIM8_OC5_COMP2 (1)(3) – LL_COMP_BLANKINGSRC_TIM15_OC1_COMP2 (1)(3)
Notes	• Availability of parameters of blanking source from timer depends on timers availability on the selected device. • Blanking source may be specific to each comparator instance. Refer to description of parameters or to reference manual.
Reference Manual to LL API cross reference:	• CSR BLANKING LL_COMP_GetOutputBlankingSource

LL_COMP_SetInputNonInverting

Function name	**__STATIC_INLINE void LL_COMP_SetInputNonInverting (COMP_TypeDef * COMPx, uint32_t InputNonInverting)**
Function description	

LL_COMP_GetInputNonInverting

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetInputNonInverting (COMP_TypeDef * COMPx)**
Function description	

LL_COMP_SetInputInverting

Function name	**__STATIC_INLINE void LL_COMP_SetInputInverting (COMP_TypeDef * COMPx, uint32_t InputInverting)**
Function description	

LL_COMP_GetInputInverting

Function name	**__STATIC_INLINE uint32_t LL_COMP_GetInputInverting (COMP_TypeDef * COMPx)**
Function description	

LL_COMP_Enable

Function name	**__STATIC_INLINE void LL_COMP_Enable (COMP_TypeDef * COMPx)**
Function description	Enable comparator instance.
Parameters	• **COMPx:** Comparator instance
Return values	• **None:**
Notes	• After enable from off state, comparator requires a delay to reach reach propagation delay specification. Refer to device datasheet, parameter "tSTART".
Reference Manual to LL API cross reference:	• CSR EN LL_COMP_Enable

LL_COMP_Disable

Function name	**__STATIC_INLINE void LL_COMP_Disable (COMP_TypeDef * COMPx)**
Function description	Disable comparator instance.
Parameters	• **COMPx:** Comparator instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR EN LL_COMP_Disable

LL_COMP_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_COMP_IsEnabled (COMP_TypeDef * COMPx)**
Function description	Get comparator enable state (0: COMP is disabled, 1: COMP is enabled)
Parameters	• **COMPx:** Comparator instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CSR EN LL_COMP_IsEnabled

reference:

LL_COMP_Lock

Function name	**__STATIC_INLINE void LL_COMP_Lock (COMP_TypeDef * COMPx)**
Function description	Lock comparator instance.
Parameters	• **COMPx:** Comparator instance
Return values	• **None:**
Notes	• Once locked, comparator configuration can be accessed in read-only. • The only way to unlock the comparator is a device hardware reset.
Reference Manual to LL API cross reference:	• CSR LOCK LL_COMP_Lock

LL_COMP_IsLocked

Function name	**__STATIC_INLINE uint32_t LL_COMP_IsLocked (COMP_TypeDef * COMPx)**
Function description	Get comparator lock state (0: COMP is unlocked, 1: COMP is locked).
Parameters	• **COMPx:** Comparator instance
Return values	• **State:** of bit (1 or 0).
Notes	• Once locked, comparator configuration can be accessed in read-only. • The only way to unlock the comparator is a device hardware reset.
Reference Manual to LL API cross reference:	• CSR LOCK LL_COMP_IsLocked

LL_COMP_ReadOutputLevel

Function name	**__STATIC_INLINE uint32_t LL_COMP_ReadOutputLevel (COMP_TypeDef * COMPx)**
Function description	Read comparator instance output level.
Parameters	• **COMPx:** Comparator instance
Return values	• **Returned:** value can be one of the following values: – LL_COMP_OUTPUT_LEVEL_LOW – LL_COMP_OUTPUT_LEVEL_HIGH
Notes	• The comparator output level depends on the selected polarity (Refer to function LL_COMP_SetOutputPolarity()). If the comparator polarity is not inverted: Comparator output is low when the input plus is at a lower voltage than the input minusComparator output is high when the input plus is at a

	higher voltage than the input minus If the comparator polarity is inverted:Comparator output is high when the input plus is at a lower voltage than the input minusComparator output is low when the input plus is at a higher voltage than the input minus
Reference Manual to LL API cross reference:	• CSR VALUE LL_COMP_ReadOutputLevel

LL_COMP_DeInit

Function name	**ErrorStatus LL_COMP_DeInit (COMP_TypeDef * COMPx)**
Function description	De-initialize registers of the selected COMP instance to their default reset values.
Parameters	• **COMPx:** COMP instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: COMP registers are de-initialized – ERROR: COMP registers are not de-initialized
Notes	• If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.

LL_COMP_Init

Function name	**ErrorStatus LL_COMP_Init (COMP_TypeDef * COMPx, LL_COMP_InitTypeDef * COMP_InitStruct)**
Function description	Initialize some features of COMP instance.
Parameters	• **COMPx:** COMP instance • **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: COMP registers are initialized – ERROR: COMP registers are not initialized
Notes	• This function configures features of the selected COMP instance. Some features are also available at scope COMP common instance (common to several COMP instances). Refer to functions having argument "COMPxy_COMMON" as parameter.

LL_COMP_StructInit

Function name	**void LL_COMP_StructInit (LL_COMP_InitTypeDef * COMP_InitStruct)**
Function description	Set each LL_COMP_InitTypeDef field to default value.
Parameters	• **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

75.3 COMP Firmware driver defines

75.3.1 COMP

Comparator common modes - Window mode

LL_COMP_WINDOWMODE_DISABLE	Window mode disable: Comparators 1 and 2 are independent
LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON	Window mode enable: Comparators instances pair COMP1 and COMP2 have their input plus connected together. The common input is COMP1 input plus (COMP2 input plus is no more accessible).

Definitions of COMP hardware constraints delays

LL_COMP_DELAY_STARTUP_US	Delay for COMP startup time
LL_COMP_DELAY_VOLTAGE_SCALER_STAB_US	Delay for COMP voltage scaler stabilization time

Comparator input - Hysteresis

LL_COMP_HYSTERESIS_NONE	No hysteresis
LL_COMP_HYSTERESIS_LOW	Hysteresis level low
LL_COMP_HYSTERESIS_MEDIUM	Hysteresis level medium
LL_COMP_HYSTERESIS_HIGH	Hysteresis level high

Comparator inputs legacy literals name

LL_COMP_WINDOWMODE_ENABLE

LL_COMP_INVERTINGINPUT_1_4VREFINT

LL_COMP_INVERTINGINPUT_1_2VREFINT

LL_COMP_INVERTINGINPUT_3_4VREFINT

LL_COMP_INVERTINGINPUT_VREFINT

LL_COMP_INVERTINGINPUT_DAC1

LL_COMP_INVERTINGINPUT_DAC2

LL_COMP_INVERTINGINPUT_IO1

LL_COMP_INVERTINGINPUT_IO2

LL_COMP_NONINVERTINGINPUT_IO1

LL_COMP_NONINVERTINGINPUT_IO2

Comparator inputs - Input minus (input inverting) selection

LL_COMP_INPUT_MINUS_1_4VREFINT	Comparator input minus connected to 1/4 VrefInt
LL_COMP_INPUT_MINUS_1_2VREFINT	Comparator input minus connected to 1/2

	VrefInt
LL_COMP_INPUT_MINUS_3_4VREFINT	Comparator input minus connected to 3/4 VrefInt
LL_COMP_INPUT_MINUS_VREFINT	Comparator input minus connected to VrefInt
LL_COMP_INPUT_MINUS_DAC1_CH1	Comparator input minus connected to DAC1 channel 1 (DAC_OUT1)
LL_COMP_INPUT_MINUS_DAC1_CH2	Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)
LL_COMP_INPUT_MINUS_IO1	Comparator input minus connected to IO1 (pin PB1 for COMP1, pin PB3 for COMP2)
LL_COMP_INPUT_MINUS_IO2	Comparator input minus connected to IO2 (pin PC4 for COMP1, pin PB7 for COMP2)

Comparator inputs - Input plus (input non-inverting) selection

LL_COMP_INPUT_PLUS_IO1	Comparator input plus connected to IO1 (pin PC5 for COMP1, pin PB4 for COMP2)
LL_COMP_INPUT_PLUS_IO2	Comparator input plus connected to IO2 (pin PB2 for COMP1, pin PB6 for COMP2)

Comparator output - Blanking source

LL_COMP_BLANKINGSRC_NONE	Comparator output without blanking
LL_COMP_BLANKINGSRC_TIM1_OC5_COMP1	Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP1)
LL_COMP_BLANKINGSRC_TIM2_OC3_COMP1	Comparator output blanking source TIM2 OC3 (specific to COMP instance: COMP1)
LL_COMP_BLANKINGSRC_TIM3_OC3_COMP1	Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP1)
LL_COMP_BLANKINGSRC_TIM3_OC4_COMP2	Comparator output blanking source TIM3 OC4 (specific to COMP instance: COMP2)
LL_COMP_BLANKINGSRC_TIM8_OC5_COMP2	Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP2)
LL_COMP_BLANKINGSRC_TIM15_OC1_COMP2	Comparator output blanking source TIM15 OC1 (specific to COMP instance: COMP2)

Comparator output blanking source legacy literals name

LL_COMP_BLANKINGSRC_TIM1_OC5

LL_COMP_BLANKINGSRC_TIM2_OC3

LL_COMP_BLANKINGSRC_TIM3_OC3

LL_COMP_BLANKINGSRC_TIM3_OC4

LL_COMP_BLANKINGSRC_TIM8_OC5

LL_COMP_BLANKINGSRC_TIM15_OC1

Comparator output - Output level

LL_COMP_OUTPUT_LEVEL_LOW	Comparator output level low (if the polarity is not inverted, otherwise to be complemented)
LL_COMP_OUTPUT_LEVEL_HIGH	Comparator output level high (if the polarity is not inverted, otherwise to be complemented)

Comparator output - Output polarity

LL_COMP_OUTPUTPOL_NONINVERTED	COMP output polarity is not inverted: comparator output is high when the plus (non-inverting) input is at a higher voltage than the minus (inverting) input
LL_COMP_OUTPUTPOL_INVERTED	COMP output polarity is inverted: comparator output is low when the plus (non-inverting) input is at a lower voltage than the minus (inverting) input

Comparator modes - Power mode

LL_COMP_POWERMODE_HIGHSPEED	COMP power mode to high speed
LL_COMP_POWERMODE_MEDIUMSPEED	COMP power mode to medium speed
LL_COMP_POWERMODE_ULTRALOWPOWER	COMP power mode to ultra-low power

COMP helper macro

__LL_COMP_COMMON_INSTANCE

Description:

- Helper macro to select the COMP common instance to which is belonging the selected COMP instance.

Parameters:

- __COMPx__: COMP instance

Return value:

- COMP: common instance or value "0" if there is no COMP common instance.

Notes:

- COMP common register instance can be used to set parameters common to several COMP instances. Refer to functions having argument "COMPxy_COMMON" as parameter.

Common write and read registers macro

LL_COMP_WriteReg

Description:

- Write a value in COMP register.

Parameters:

- __INSTANCE__: comparator instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_COMP_ReadReg

Description:

- Read a value in COMP register.

Parameters:

- __INSTANCE__: comparator instance
- __REG__: Register to be read

Return value:

- Register: value

76 LL CORTEX Generic Driver

76.1 CORTEX Firmware driver API description

76.1.1 Detailed description of functions

LL_SYSTICK_IsActiveCounterFlag

Function name	**__STATIC_INLINE uint32_t LL_SYSTICK_IsActiveCounterFlag (void)**
Function description	This function checks if the Systick counter flag is active or not.
Return values	• **State:** of bit (1 or 0).
Notes	• It can be used in timeout function on application side.
Reference Manual to LL API cross reference:	• STK_CTRL COUNTFLAG LL_SYSTICK_IsActiveCounterFlag

LL_SYSTICK_SetClkSource

Function name	**__STATIC_INLINE void LL_SYSTICK_SetClkSource (uint32_t Source)**
Function description	Configures the SysTick clock source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_SYSTICK_CLKSOURCE_HCLK_DIV8 – LL_SYSTICK_CLKSOURCE_HCLK
Return values	• **None:**
Reference Manual to LL API cross reference:	• STK_CTRL CLKSOURCE LL_SYSTICK_SetClkSource

LL_SYSTICK_GetClkSource

Function name	**__STATIC_INLINE uint32_t LL_SYSTICK_GetClkSource (void)**
Function description	Get the SysTick clock source.
Return values	• **Returned:** value can be one of the following values: – LL_SYSTICK_CLKSOURCE_HCLK_DIV8 – LL_SYSTICK_CLKSOURCE_HCLK
Reference Manual to LL API cross reference:	• STK_CTRL CLKSOURCE LL_SYSTICK_GetClkSource

LL_SYSTICK_EnableIT

Function name	**__STATIC_INLINE void LL_SYSTICK_EnableIT (void)**
Function description	Enable SysTick exception request.

Return values	• **None:**
Reference Manual to LL API cross reference:	• STK_CTRL TICKINT LL_SYSTICK_EnableIT

LL_SYSTICK_DisableIT

Function name	**__STATIC_INLINE void LL_SYSTICK_DisableIT (void)**
Function description	Disable SysTick exception request.
Return values	• **None:**
Reference Manual to LL API cross reference:	• STK_CTRL TICKINT LL_SYSTICK_DisableIT

LL_SYSTICK_IsEnabledIT

Function name	**__STATIC_INLINE uint32_t LL_SYSTICK_IsEnabledIT (void)**
Function description	Checks if the SYSTICK interrupt is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• STK_CTRL TICKINT LL_SYSTICK_IsEnabledIT

LL_LPM_EnableSleep

Function name	**__STATIC_INLINE void LL_LPM_EnableSleep (void)**
Function description	Processor uses sleep as its low power mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPDEEP LL_LPM_EnableSleep

LL_LPM_EnableDeepSleep

Function name	**__STATIC_INLINE void LL_LPM_EnableDeepSleep (void)**
Function description	Processor uses deep sleep as its low power mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPDEEP LL_LPM_EnableDeepSleep

LL_LPM_EnableSleepOnExit

Function name	**__STATIC_INLINE void LL_LPM_EnableSleepOnExit (void)**
Function description	Configures sleep-on-exit when returning from Handler mode to Thread mode.

Return values	• **None:**
Notes	• Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPONEXIT LL_LPM_EnableSleepOnExit

LL_LPM_DisableSleepOnExit

Function name	**__STATIC_INLINE void LL_LPM_DisableSleepOnExit (void)**
Function description	Do not sleep when returning to Thread mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPONEXIT LL_LPM_DisableSleepOnExit

LL_LPM_EnableEventOnPend

Function name	**__STATIC_INLINE void LL_LPM_EnableEventOnPend (void)**
Function description	Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SCR SEVEONPEND LL_LPM_EnableEventOnPend

LL_LPM_DisableEventOnPend

Function name	**__STATIC_INLINE void LL_LPM_DisableEventOnPend (void)**
Function description	Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SCR SEVEONPEND LL_LPM_DisableEventOnPend

LL_HANDLER_EnableFault

Function name	**__STATIC_INLINE void LL_HANDLER_EnableFault (uint32_t Fault)**
Function description	Enable a fault in System handler control register (SHCSR)
Parameters	• **Fault:** This parameter can be a combination of the following values: – LL_HANDLER_FAULT_USG – LL_HANDLER_FAULT_BUS – LL_HANDLER_FAULT_MEM

Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SHCSR MEMFAULTENA LL_HANDLER_EnableFault

LL_HANDLER_DisableFault

Function name	**__STATIC_INLINE void LL_HANDLER_DisableFault (uint32_t Fault)**
Function description	Disable a fault in System handler control register (SHCSR)
Parameters	• **Fault:** This parameter can be a combination of the following values: – LL_HANDLER_FAULT_USG – LL_HANDLER_FAULT_BUS – LL_HANDLER_FAULT_MEM
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SHCSR MEMFAULTENA LL_HANDLER_DisableFault

LL_CPUID_GetImplementer

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetImplementer (void)**
Function description	Get Implementer code.
Return values	• **Value:** should be equal to 0x41 for Arm
Reference Manual to LL API cross reference:	• SCB_CPUID IMPLEMENTER LL_CPUID_GetImplementer

LL_CPUID_GetVariant

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetVariant (void)**
Function description	Get Variant number (The r value in the rnpn product revision identifier)
Return values	• **Value:** between 0 and 255 (0x0: revision 0)
Reference Manual to LL API cross reference:	• SCB_CPUID VARIANT LL_CPUID_GetVariant

LL_CPUID_GetConstant

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetConstant (void)**
Function description	Get Constant number.
Return values	• **Value:** should be equal to 0xF for Cortex-M4 devices
Reference Manual to LL API cross	• SCB_CPUID ARCHITECTURE LL_CPUID_GetConstant

reference:

LL_CPUID_GetParNo

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetParNo (void)**
Function description	Get Part number.
Return values	• **Value:** should be equal to 0xC24 for Cortex-M4
Reference Manual to LL API cross reference:	• SCB_CPUID PARTNO LL_CPUID_GetParNo

LL_CPUID_GetRevision

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetRevision (void)**
Function description	Get Revision number (The p value in the rnpn product revision identifier, indicates patch release)
Return values	• **Value:** between 0 and 255 (0x1: patch 1)
Reference Manual to LL API cross reference:	• SCB_CPUID REVISION LL_CPUID_GetRevision

LL_MPU_Enable

Function name	**__STATIC_INLINE void LL_MPU_Enable (uint32_t Options)**
Function description	Enable MPU with input options.
Parameters	• **Options:** This parameter can be one of the following values: – LL_MPU_CTRL_HFNMI_PRIVDEF_NONE – LL_MPU_CTRL_HARDFAULT_NMI – LL_MPU_CTRL_PRIVILEGED_DEFAULT – LL_MPU_CTRL_HFNMI_PRIVDEF
Return values	• **None:**
Reference Manual to LL API cross reference:	• MPU_CTRL ENABLE LL_MPU_Enable

LL_MPU_Disable

Function name	**__STATIC_INLINE void LL_MPU_Disable (void)**
Function description	Disable MPU.
Return values	• **None:**
Reference Manual to LL API cross reference:	• MPU_CTRL ENABLE LL_MPU_Disable

LL_MPU_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_MPU_IsEnabled (void)**

Function description	Check if MPU is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MPU_CTRL ENABLE LL_MPU_IsEnabled

LL_MPU_EnableRegion

Function name	**__STATIC_INLINE void LL_MPU_EnableRegion (uint32_t Region)**
Function description	Enable a MPU region.
Parameters	• **Region:** This parameter can be one of the following values: – LL_MPU_REGION_NUMBER0 – LL_MPU_REGION_NUMBER1 – LL_MPU_REGION_NUMBER2 – LL_MPU_REGION_NUMBER3 – LL_MPU_REGION_NUMBER4 – LL_MPU_REGION_NUMBER5 – LL_MPU_REGION_NUMBER6 – LL_MPU_REGION_NUMBER7
Return values	• **None:**
Reference Manual to LL API cross reference:	• MPU_RASR ENABLE LL_MPU_EnableRegion

LL_MPU_ConfigRegion

Function name	**__STATIC_INLINE void LL_MPU_ConfigRegion (uint32_t Region, uint32_t SubRegionDisable, uint32_t Address, uint32_t Attributes)**
Function description	Configure and enable a region.
Parameters	• **Region:** This parameter can be one of the following values: – LL_MPU_REGION_NUMBER0 – LL_MPU_REGION_NUMBER1 – LL_MPU_REGION_NUMBER2 – LL_MPU_REGION_NUMBER3 – LL_MPU_REGION_NUMBER4 – LL_MPU_REGION_NUMBER5 – LL_MPU_REGION_NUMBER6 – LL_MPU_REGION_NUMBER7 • **Address:** Value of region base address • **SubRegionDisable:** Sub-region disable value between Min_Data = 0x00 and Max_Data = 0xFF • **Attributes:** This parameter can be a combination of the following values: – LL_MPU_REGION_SIZE_32B or LL_MPU_REGION_SIZE_64B or LL_MPU_REGION_SIZE_128B or LL_MPU_REGION_SIZE_256B or LL_MPU_REGION_SIZE_512B or

LL_MPU_REGION_SIZE_1KB or
LL_MPU_REGION_SIZE_2KB or
LL_MPU_REGION_SIZE_4KB or
LL_MPU_REGION_SIZE_8KB or
LL_MPU_REGION_SIZE_16KB or
LL_MPU_REGION_SIZE_32KB or
LL_MPU_REGION_SIZE_64KB or
LL_MPU_REGION_SIZE_128KB or
LL_MPU_REGION_SIZE_256KB or
LL_MPU_REGION_SIZE_512KB or
LL_MPU_REGION_SIZE_1MB or
LL_MPU_REGION_SIZE_2MB or
LL_MPU_REGION_SIZE_4MB or
LL_MPU_REGION_SIZE_8MB or
LL_MPU_REGION_SIZE_16MB or
LL_MPU_REGION_SIZE_32MB or
LL_MPU_REGION_SIZE_64MB or
LL_MPU_REGION_SIZE_128MB or
LL_MPU_REGION_SIZE_256MB or
LL_MPU_REGION_SIZE_512MB or
LL_MPU_REGION_SIZE_1GB or
LL_MPU_REGION_SIZE_2GB or
LL_MPU_REGION_SIZE_4GB

- LL_MPU_REGION_NO_ACCESS or LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_PRIV_RW_URO or LL_MPU_REGION_FULL_ACCESS or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_PRIV_RO_URO
- LL_MPU_TEX_LEVEL0 or LL_MPU_TEX_LEVEL1 or LL_MPU_TEX_LEVEL2 or LL_MPU_TEX_LEVEL4
- LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
- LL_MPU_ACCESS_SHAREABLE or LL_MPU_ACCESS_NOT_SHAREABLE
- LL_MPU_ACCESS_CACHEABLE or LL_MPU_ACCESS_NOT_CACHEABLE
- LL_MPU_ACCESS_BUFFERABLE or LL_MPU_ACCESS_NOT_BUFFERABLE

Return values	• **None:**
Reference Manual to LL API cross reference:	• MPU_RNR REGION LL_MPU_ConfigRegion • MPU_RBAR REGION LL_MPU_ConfigRegion • MPU_RBAR ADDR LL_MPU_ConfigRegion • MPU_RASR XN LL_MPU_ConfigRegion • MPU_RASR AP LL_MPU_ConfigRegion • MPU_RASR S LL_MPU_ConfigRegion • MPU_RASR C LL_MPU_ConfigRegion • MPU_RASR B LL_MPU_ConfigRegion • MPU_RASR SIZE LL_MPU_ConfigRegion

LL_MPU_DisableRegion

Function name	**__STATIC_INLINE void LL_MPU_DisableRegion (uint32_t**

	Region)
Function description	Disable a region.
Parameters	• **Region:** This parameter can be one of the following values: – LL_MPU_REGION_NUMBER0 – LL_MPU_REGION_NUMBER1 – LL_MPU_REGION_NUMBER2 – LL_MPU_REGION_NUMBER3 – LL_MPU_REGION_NUMBER4 – LL_MPU_REGION_NUMBER5 – LL_MPU_REGION_NUMBER6 – LL_MPU_REGION_NUMBER7
Return values	• **None:**
Reference Manual to LL API cross reference:	• MPU_RNR REGION LL_MPU_DisableRegion • MPU_RASR ENABLE LL_MPU_DisableRegion

76.2 CORTEX Firmware driver defines

76.2.1 CORTEX

MPU Bufferable Access

LL_MPU_ACCESS_BUFFERABLE	Bufferable memory attribute
LL_MPU_ACCESS_NOT_BUFFERABLE	Not Bufferable memory attribute

MPU Cacheable Access

LL_MPU_ACCESS_CACHEABLE	Cacheable memory attribute
LL_MPU_ACCESS_NOT_CACHEABLE	Not Cacheable memory attribute

SYSTICK Clock Source

LL_SYSTICK_CLKSOURCE_HCLK_DIV8	AHB clock divided by 8 selected as SysTick clock source.
LL_SYSTICK_CLKSOURCE_HCLK	AHB clock selected as SysTick clock source.

MPU Control

LL_MPU_CTRL_HFNMI_PRIVDEF_NONE	Disable NMI and privileged SW access
LL_MPU_CTRL_HARDFAULT_NMI	Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers
LL_MPU_CTRL_PRIVILEGED_DEFAULT	Enable privileged software access to default memory map
LL_MPU_CTRL_HFNMI_PRIVDEF	Enable NMI and privileged SW access

Handler Fault type

LL_HANDLER_FAULT_USG	Usage fault
LL_HANDLER_FAULT_BUS	Bus fault
LL_HANDLER_FAULT_MEM	Memory management fault

MPU Instruction Access

LL_MPU_INSTRUCTION_ACCESS_ENABLE	Instruction fetches enabled
LL_MPU_INSTRUCTION_ACCESS_DISABLE	Instruction fetches disabled

MPU Region Number

LL_MPU_REGION_NUMBER0	REGION Number 0
LL_MPU_REGION_NUMBER1	REGION Number 1
LL_MPU_REGION_NUMBER2	REGION Number 2
LL_MPU_REGION_NUMBER3	REGION Number 3
LL_MPU_REGION_NUMBER4	REGION Number 4
LL_MPU_REGION_NUMBER5	REGION Number 5
LL_MPU_REGION_NUMBER6	REGION Number 6
LL_MPU_REGION_NUMBER7	REGION Number 7

MPU Region Privileges

LL_MPU_REGION_NO_ACCESS	No access
LL_MPU_REGION_PRIV_RW	RW privileged (privileged access only)
LL_MPU_REGION_PRIV_RW_URO	RW privileged - RO user (Write in a user program generates a fault)
LL_MPU_REGION_FULL_ACCESS	RW privileged & user (Full access)
LL_MPU_REGION_PRIV_RO	RO privileged (privileged read only)
LL_MPU_REGION_PRIV_RO_URO	RO privileged & user (read only)

MPU Region Size

LL_MPU_REGION_SIZE_32B	32B Size of the MPU protection region
LL_MPU_REGION_SIZE_64B	64B Size of the MPU protection region
LL_MPU_REGION_SIZE_128B	128B Size of the MPU protection region
LL_MPU_REGION_SIZE_256B	256B Size of the MPU protection region
LL_MPU_REGION_SIZE_512B	512B Size of the MPU protection region
LL_MPU_REGION_SIZE_1KB	1KB Size of the MPU protection region
LL_MPU_REGION_SIZE_2KB	2KB Size of the MPU protection region
LL_MPU_REGION_SIZE_4KB	4KB Size of the MPU protection region
LL_MPU_REGION_SIZE_8KB	8KB Size of the MPU protection region
LL_MPU_REGION_SIZE_16KB	16KB Size of the MPU protection region
LL_MPU_REGION_SIZE_32KB	32KB Size of the MPU protection region
LL_MPU_REGION_SIZE_64KB	64KB Size of the MPU protection region
LL_MPU_REGION_SIZE_128KB	128KB Size of the MPU protection region
LL_MPU_REGION_SIZE_256KB	256KB Size of the MPU protection region
LL_MPU_REGION_SIZE_512KB	512KB Size of the MPU protection region
LL_MPU_REGION_SIZE_1MB	1MB Size of the MPU protection region
LL_MPU_REGION_SIZE_2MB	2MB Size of the MPU protection region

LL_MPU_REGION_SIZE_4MB	4MB Size of the MPU protection region
LL_MPU_REGION_SIZE_8MB	8MB Size of the MPU protection region
LL_MPU_REGION_SIZE_16MB	16MB Size of the MPU protection region
LL_MPU_REGION_SIZE_32MB	32MB Size of the MPU protection region
LL_MPU_REGION_SIZE_64MB	64MB Size of the MPU protection region
LL_MPU_REGION_SIZE_128MB	128MB Size of the MPU protection region
LL_MPU_REGION_SIZE_256MB	256MB Size of the MPU protection region
LL_MPU_REGION_SIZE_512MB	512MB Size of the MPU protection region
LL_MPU_REGION_SIZE_1GB	1GB Size of the MPU protection region
LL_MPU_REGION_SIZE_2GB	2GB Size of the MPU protection region
LL_MPU_REGION_SIZE_4GB	4GB Size of the MPU protection region

MPU Shareable Access

LL_MPU_ACCESS_SHAREABLE	Shareable memory attribute
LL_MPU_ACCESS_NOT_SHAREABLE	Not Shareable memory attribute

MPU TEX Level

LL_MPU_TEX_LEVEL0	b000 for TEX bits
LL_MPU_TEX_LEVEL1	b001 for TEX bits
LL_MPU_TEX_LEVEL2	b010 for TEX bits
LL_MPU_TEX_LEVEL4	b100 for TEX bits

77 LL CRC Generic Driver

77.1 CRC Firmware driver API description

77.1.1 Detailed description of functions

LL_CRC_ResetCRCCalculationUnit

Function name	**__STATIC_INLINE void LL_CRC_ResetCRCCalculationUnit (CRC_TypeDef * CRCx)**
Function description	Reset the CRC calculation unit.
Parameters	• **CRCx:** CRC Instance
Return values	• **None:**
Notes	• If Programmable Initial CRC value feature is available, also set the Data Register to the value stored in the CRC_INIT register, otherwise, reset Data Register to its default value.
Reference Manual to LL API cross reference:	• CR RESET LL_CRC_ResetCRCCalculationUnit

LL_CRC_SetPolynomialSize

Function name	**__STATIC_INLINE void LL_CRC_SetPolynomialSize (CRC_TypeDef * CRCx, uint32_t PolySize)**
Function description	Configure size of the polynomial.
Parameters	• **CRCx:** CRC Instance • **PolySize:** This parameter can be one of the following values: – LL_CRC_POLYLENGTH_32B – LL_CRC_POLYLENGTH_16B – LL_CRC_POLYLENGTH_8B – LL_CRC_POLYLENGTH_7B
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR POLYSIZE LL_CRC_SetPolynomialSize

LL_CRC_GetPolynomialSize

Function name	**__STATIC_INLINE uint32_t LL_CRC_GetPolynomialSize (CRC_TypeDef * CRCx)**
Function description	Return size of the polynomial.
Parameters	• **CRCx:** CRC Instance
Return values	• **Returned:** value can be one of the following values: – LL_CRC_POLYLENGTH_32B – LL_CRC_POLYLENGTH_16B

– LL_CRC_POLYLENGTH_8B
– LL_CRC_POLYLENGTH_7B

Reference Manual to LL API cross reference:
- CR POLYSIZE LL_CRC_GetPolynomialSize

LL_CRC_SetInputDataReverseMode

Function name
__STATIC_INLINE void LL_CRC_SetInputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description
Configure the reversal of the bit order of the input data.

Parameters
- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  – LL_CRC_INDATA_REVERSE_NONE
  – LL_CRC_INDATA_REVERSE_BYTE
  – LL_CRC_INDATA_REVERSE_HALFWORD
  – LL_CRC_INDATA_REVERSE_WORD

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR REV_IN LL_CRC_SetInputDataReverseMode

LL_CRC_GetInputDataReverseMode

Function name
__STATIC_INLINE uint32_t LL_CRC_GetInputDataReverseMode (CRC_TypeDef * CRCx)

Function description
Return type of reversal for input data bit order.

Parameters
- **CRCx:** CRC Instance

Return values
- **Returned:** value can be one of the following values:
  – LL_CRC_INDATA_REVERSE_NONE
  – LL_CRC_INDATA_REVERSE_BYTE
  – LL_CRC_INDATA_REVERSE_HALFWORD
  – LL_CRC_INDATA_REVERSE_WORD

Reference Manual to LL API cross reference:
- CR REV_IN LL_CRC_GetInputDataReverseMode

LL_CRC_SetOutputDataReverseMode

Function name
__STATIC_INLINE void LL_CRC_SetOutputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description
Configure the reversal of the bit order of the Output data.

Parameters
- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  – LL_CRC_OUTDATA_REVERSE_NONE
  – LL_CRC_OUTDATA_REVERSE_BIT

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR REV_OUT LL_CRC_SetOutputDataReverseMode

LL_CRC_GetOutputDataReverseMode

Function name	**__STATIC_INLINE uint32_t LL_CRC_GetOutputDataReverseMode (CRC_TypeDef * CRCx)**
Function description	Configure the reversal of the bit order of the Output data.
Parameters	• **CRCx:** CRC Instance
Return values	• **Returned:** value can be one of the following values: – LL_CRC_OUTDATA_REVERSE_NONE – LL_CRC_OUTDATA_REVERSE_BIT
Reference Manual to LL API cross reference:	• CR REV_OUT LL_CRC_GetOutputDataReverseMode

LL_CRC_SetInitialData

Function name	**__STATIC_INLINE void LL_CRC_SetInitialData (CRC_TypeDef * CRCx, uint32_t InitCrc)**
Function description	Initialize the Programmable initial CRC value.
Parameters	• **CRCx:** CRC Instance • **InitCrc:** Value to be programmed in Programmable initial CRC value register
Return values	• **None:**
Notes	• If the CRC size is less than 32 bits, the least significant bits are used to write the correct value • LL_CRC_DEFAULT_CRC_INITVALUE could be used as value for InitCrc parameter.
Reference Manual to LL API cross reference:	• INIT INIT LL_CRC_SetInitialData

LL_CRC_GetInitialData

Function name	**__STATIC_INLINE uint32_t LL_CRC_GetInitialData (CRC_TypeDef * CRCx)**
Function description	Return current Initial CRC value.
Parameters	• **CRCx:** CRC Instance
Return values	• **Value:** programmed in Programmable initial CRC value register
Notes	• If the CRC size is less than 32 bits, the least significant bits are used to read the correct value
Reference Manual to	• INIT INIT LL_CRC_GetInitialData

LL API cross reference:

LL_CRC_SetPolynomialCoef

Function name	**__STATIC_INLINE void LL_CRC_SetPolynomialCoef (CRC_TypeDef * CRCx, uint32_t PolynomCoef)**
Function description	Initialize the Programmable polynomial value (coefficients of the polynomial to be used for CRC calculation).
Parameters	• **CRCx:** CRC Instance • **PolynomCoef:** Value to be programmed in Programmable Polynomial value register
Return values	• **None:**
Notes	• LL_CRC_DEFAULT_CRC32_POLY could be used as value for PolynomCoef parameter. • Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65
Reference Manual to LL API cross reference:	• POL POL LL_CRC_SetPolynomialCoef

LL_CRC_GetPolynomialCoef

Function name	**__STATIC_INLINE uint32_t LL_CRC_GetPolynomialCoef (CRC_TypeDef * CRCx)**
Function description	Return current Programmable polynomial value.
Parameters	• **CRCx:** CRC Instance
Return values	• **Value:** programmed in Programmable Polynomial value register
Notes	• Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65
Reference Manual to LL API cross reference:	• POL POL LL_CRC_GetPolynomialCoef

LL_CRC_FeedData32

Function name	**__STATIC_INLINE void LL_CRC_FeedData32 (CRC_TypeDef * CRCx, uint32_t InData)**
Function description	Write given 32-bit data to the CRC calculator.
Parameters	• **CRCx:** CRC Instance • **InData:** value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_CRC_FeedData32

LL_CRC_FeedData16

Function name	**__STATIC_INLINE void LL_CRC_FeedData16 (CRC_TypeDef * CRCx, uint16_t InData)**
Function description	Write given 16-bit data to the CRC calculator.
Parameters	• **CRCx:** CRC Instance • **InData:** 16 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_CRC_FeedData16

LL_CRC_FeedData8

Function name	**__STATIC_INLINE void LL_CRC_FeedData8 (CRC_TypeDef * CRCx, uint8_t InData)**
Function description	Write given 8-bit data to the CRC calculator.
Parameters	• **CRCx:** CRC Instance • **InData:** 8 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_CRC_FeedData8

LL_CRC_ReadData32

Function name	**__STATIC_INLINE uint32_t LL_CRC_ReadData32 (CRC_TypeDef * CRCx)**
Function description	Return current CRC calculation result.
Parameters	• **CRCx:** CRC Instance
Return values	• **Current:** CRC calculation result as stored in CRC_DR register (32 bits).
Reference Manual to LL API cross reference:	• DR DR LL_CRC_ReadData32

LL_CRC_ReadData16

Function name	**__STATIC_INLINE uint16_t LL_CRC_ReadData16 (CRC_TypeDef * CRCx)**

Function description	Return current CRC calculation result.
Parameters	• **CRCx:** CRC Instance
Return values	• **Current:** CRC calculation result as stored in CRC_DR register (16 bits).
Notes	• This function is expected to be used in a 16 bits CRC polynomial size context.
Reference Manual to LL API cross reference:	• DR DR LL_CRC_ReadData16

LL_CRC_ReadData8

Function name	**__STATIC_INLINE uint8_t LL_CRC_ReadData8 (CRC_TypeDef * CRCx)**
Function description	Return current CRC calculation result.
Parameters	• **CRCx:** CRC Instance
Return values	• **Current:** CRC calculation result as stored in CRC_DR register (8 bits).
Notes	• This function is expected to be used in a 8 bits CRC polynomial size context.
Reference Manual to LL API cross reference:	• DR DR LL_CRC_ReadData8

LL_CRC_ReadData7

Function name	**__STATIC_INLINE uint8_t LL_CRC_ReadData7 (CRC_TypeDef * CRCx)**
Function description	Return current CRC calculation result.
Parameters	• **CRCx:** CRC Instance
Return values	• **Current:** CRC calculation result as stored in CRC_DR register (7 bits).
Notes	• This function is expected to be used in a 7 bits CRC polynomial size context.
Reference Manual to LL API cross reference:	• DR DR LL_CRC_ReadData7

LL_CRC_Read_IDR

Function name	**__STATIC_INLINE uint32_t LL_CRC_Read_IDR (CRC_TypeDef * CRCx)**
Function description	Return data stored in the Independent Data(IDR) register.
Parameters	• **CRCx:** CRC Instance
Return values	• **Value:** stored in CRC_IDR register (General-purpose 8-bit

data register).

Notes	• This register can be used as a temporary storage location for one byte.
Reference Manual to LL API cross reference:	• IDR IDR LL_CRC_Read_IDR

LL_CRC_Write_IDR

Function name	**__STATIC_INLINE void LL_CRC_Write_IDR (CRC_TypeDef * CRCx, uint32_t InData)**
Function description	Store data in the Independent Data(IDR) register.
Parameters	• **CRCx:** CRC Instance • **InData:** value to be stored in CRC_IDR register (8-bit) between between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Notes	• This register can be used as a temporary storage location for one byte.
Reference Manual to LL API cross reference:	• IDR IDR LL_CRC_Write_IDR

LL_CRC_DeInit

Function name	**ErrorStatus LL_CRC_DeInit (CRC_TypeDef * CRCx)**
Function description	De-initialize CRC registers (Registers restored to their default values).
Parameters	• **CRCx:** CRC Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: CRC registers are de-initialized – ERROR: CRC registers are not de-initialized

77.2 CRC Firmware driver defines

77.2.1 CRC

Default CRC computation initialization value

LL_CRC_DEFAULT_CRC_INITVALUE	Default CRC computation initialization value

Default CRC generating polynomial value

LL_CRC_DEFAULT_CRC32_POLY	Default CRC generating polynomial value

Input Data Reverse

LL_CRC_INDATA_REVERSE_NONE	Input Data bit order not affected
LL_CRC_INDATA_REVERSE_BYTE	Input Data bit reversal done by byte
LL_CRC_INDATA_REVERSE_HALFWORD	Input Data bit reversal done by half-word

LL_CRC_INDATA_REVERSE_WORD	Input Data bit reversal done by word

Output Data Reverse

LL_CRC_OUTDATA_REVERSE_NONE	Output Data bit order not affected
LL_CRC_OUTDATA_REVERSE_BIT	Output Data bit reversal done by bit

Polynomial length

LL_CRC_POLYLENGTH_32B	32 bits Polynomial size
LL_CRC_POLYLENGTH_16B	16 bits Polynomial size
LL_CRC_POLYLENGTH_8B	8 bits Polynomial size
LL_CRC_POLYLENGTH_7B	7 bits Polynomial size

Common Write and read registers Macros

LL_CRC_WriteReg

Description:

- Write a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRC_ReadReg

Description:

- Read a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be read

Return value:

- Register: value

78 LL CRS Generic Driver

78.1 CRS Firmware driver API description

78.1.1 Detailed description of functions

LL_CRS_EnableFreqErrorCounter

Function name	**__STATIC_INLINE void LL_CRS_EnableFreqErrorCounter (void)**
Function description	Enable Frequency error counter.
Return values	• **None:**
Notes	• When this bit is set, the CRS_CFGR register is write-protected and cannot be modified
Reference Manual to LL API cross reference:	• CR CEN LL_CRS_EnableFreqErrorCounter

LL_CRS_DisableFreqErrorCounter

Function name	**__STATIC_INLINE void LL_CRS_DisableFreqErrorCounter (void)**
Function description	Disable Frequency error counter.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CEN LL_CRS_DisableFreqErrorCounter

LL_CRS_IsEnabledFreqErrorCounter

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledFreqErrorCounter (void)**
Function description	Check if Frequency error counter is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR CEN LL_CRS_IsEnabledFreqErrorCounter

LL_CRS_EnableAutoTrimming

Function name	**__STATIC_INLINE void LL_CRS_EnableAutoTrimming (void)**
Function description	Enable Automatic trimming counter.
Return values	• **None:**
Reference Manual to LL API cross	• CR AUTOTRIMEN LL_CRS_EnableAutoTrimming

reference:

LL_CRS_DisableAutoTrimming

Function name	**__STATIC_INLINE void LL_CRS_DisableAutoTrimming (void)**
Function description	Disable Automatic trimming counter.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR AUTOTRIMEN LL_CRS_DisableAutoTrimming

LL_CRS_IsEnabledAutoTrimming

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledAutoTrimming (void)**
Function description	Check if Automatic trimming is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR AUTOTRIMEN LL_CRS_IsEnabledAutoTrimming

LL_CRS_SetHSI48SmoothTrimming

Function name	**__STATIC_INLINE void LL_CRS_SetHSI48SmoothTrimming (uint32_t Value)**
Function description	Set HSI48 oscillator smooth trimming.
Parameters	• **Value:** a number between Min_Data = 0 and Max_Data = 63
Return values	• **None:**
Notes	• When the AUTOTRIMEN bit is set, this field is controlled by hardware and is read-only • Default value can be set thanks to LL_CRS_HSI48CALIBRATION_DEFAULT
Reference Manual to LL API cross reference:	• CR TRIM LL_CRS_SetHSI48SmoothTrimming

LL_CRS_GetHSI48SmoothTrimming

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetHSI48SmoothTrimming (void)**
Function description	Get HSI48 oscillator smooth trimming.
Return values	• **a:** number between Min_Data = 0 and Max_Data = 63
Reference Manual to LL API cross reference:	• CR TRIM LL_CRS_GetHSI48SmoothTrimming

LL_CRS_SetReloadCounter

Function name	**__STATIC_INLINE void LL_CRS_SetReloadCounter (uint32_t Value)**
Function description	Set counter reload value.
Parameters	• **Value:** a number between Min_Data = 0 and Max_Data = 0xFFFF
Return values	• **None:**
Notes	• Default value can be set thanks to LL_CRS_RELOADVALUE_DEFAULT Otherwise it can be calculated in using macro __LL_CRS_CALC_CALCULATE_RELOADVALUE (_FTARGET_, _FSYNC_)
Reference Manual to LL API cross reference:	• CFGR RELOAD LL_CRS_SetReloadCounter

LL_CRS_GetReloadCounter

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetReloadCounter (void)**
Function description	Get counter reload value.
Return values	• **a:** number between Min_Data = 0 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• CFGR RELOAD LL_CRS_GetReloadCounter

LL_CRS_SetFreqErrorLimit

Function name	**__STATIC_INLINE void LL_CRS_SetFreqErrorLimit (uint32_t Value)**
Function description	Set frequency error limit.
Parameters	• **Value:** a number between Min_Data = 0 and Max_Data = 255
Return values	• **None:**
Notes	• Default value can be set thanks to LL_CRS_ERRORLIMIT_DEFAULT
Reference Manual to LL API cross reference:	• CFGR FELIM LL_CRS_SetFreqErrorLimit

LL_CRS_GetFreqErrorLimit

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorLimit (void)**
Function description	Get frequency error limit.
Return values	• **A:** number between Min_Data = 0 and Max_Data = 255
Reference Manual to LL API cross	• CFGR FELIM LL_CRS_GetFreqErrorLimit

reference:

LL_CRS_SetSyncDivider

Function name	**__STATIC_INLINE void LL_CRS_SetSyncDivider (uint32_t Divider)**
Function description	Set division factor for SYNC signal.
Parameters	• **Divider:** This parameter can be one of the following values: – LL_CRS_SYNC_DIV_1 – LL_CRS_SYNC_DIV_2 – LL_CRS_SYNC_DIV_4 – LL_CRS_SYNC_DIV_8 – LL_CRS_SYNC_DIV_16 – LL_CRS_SYNC_DIV_32 – LL_CRS_SYNC_DIV_64 – LL_CRS_SYNC_DIV_128
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR SYNCDIV LL_CRS_SetSyncDivider

LL_CRS_GetSyncDivider

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetSyncDivider (void)**
Function description	Get division factor for SYNC signal.
Return values	• **Returned:** value can be one of the following values: – LL_CRS_SYNC_DIV_1 – LL_CRS_SYNC_DIV_2 – LL_CRS_SYNC_DIV_4 – LL_CRS_SYNC_DIV_8 – LL_CRS_SYNC_DIV_16 – LL_CRS_SYNC_DIV_32 – LL_CRS_SYNC_DIV_64 – LL_CRS_SYNC_DIV_128
Reference Manual to LL API cross reference:	• CFGR SYNCDIV LL_CRS_GetSyncDivider

LL_CRS_SetSyncSignalSource

Function name	**__STATIC_INLINE void LL_CRS_SetSyncSignalSource (uint32_t Source)**
Function description	Set SYNC signal source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_CRS_SYNC_SOURCE_GPIO – LL_CRS_SYNC_SOURCE_LSE – LL_CRS_SYNC_SOURCE_USB
Return values	• **None:**

Reference Manual to LL API cross reference:	• CFGR SYNCSRC LL_CRS_SetSyncSignalSource

LL_CRS_GetSyncSignalSource

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetSyncSignalSource (void)**
Function description	Get SYNC signal source.
Return values	• **Returned:** value can be one of the following values: – LL_CRS_SYNC_SOURCE_GPIO – LL_CRS_SYNC_SOURCE_LSE – LL_CRS_SYNC_SOURCE_USB
Reference Manual to LL API cross reference:	• CFGR SYNCSRC LL_CRS_GetSyncSignalSource

LL_CRS_SetSyncPolarity

Function name	**__STATIC_INLINE void LL_CRS_SetSyncPolarity (uint32_t Polarity)**
Function description	Set input polarity for the SYNC signal source.
Parameters	• **Polarity:** This parameter can be one of the following values: – LL_CRS_SYNC_POLARITY_RISING – LL_CRS_SYNC_POLARITY_FALLING
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR SYNCPOL LL_CRS_SetSyncPolarity

LL_CRS_GetSyncPolarity

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetSyncPolarity (void)**
Function description	Get input polarity for the SYNC signal source.
Return values	• **Returned:** value can be one of the following values: – LL_CRS_SYNC_POLARITY_RISING – LL_CRS_SYNC_POLARITY_FALLING
Reference Manual to LL API cross reference:	• CFGR SYNCPOL LL_CRS_GetSyncPolarity

LL_CRS_ConfigSynchronization

Function name	**__STATIC_INLINE void LL_CRS_ConfigSynchronization (uint32_t HSI48CalibrationValue, uint32_t ErrorLimitValue, uint32_t ReloadValue, uint32_t Settings)**
Function description	Configure CRS for the synchronization.

Parameters	• **HSI48CalibrationValue:** a number between Min_Data = 0 and Max_Data = 63 • **ErrorLimitValue:** a number between Min_Data = 0 and Max_Data = 0xFFFF • **ReloadValue:** a number between Min_Data = 0 and Max_Data = 255 • **Settings:** This parameter can be a combination of the following values: – LL_CRS_SYNC_DIV_1 or LL_CRS_SYNC_DIV_2 or LL_CRS_SYNC_DIV_4 or LL_CRS_SYNC_DIV_8 or LL_CRS_SYNC_DIV_16 or LL_CRS_SYNC_DIV_32 or LL_CRS_SYNC_DIV_64 or LL_CRS_SYNC_DIV_128 – LL_CRS_SYNC_SOURCE_GPIO or LL_CRS_SYNC_SOURCE_LSE or LL_CRS_SYNC_SOURCE_USB – LL_CRS_SYNC_POLARITY_RISING or LL_CRS_SYNC_POLARITY_FALLING
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR TRIM LL_CRS_ConfigSynchronization • CFGR RELOAD LL_CRS_ConfigSynchronization • CFGR FELIM LL_CRS_ConfigSynchronization • CFGR SYNCDIV LL_CRS_ConfigSynchronization • CFGR SYNCSRC LL_CRS_ConfigSynchronization • CFGR SYNCPOL LL_CRS_ConfigSynchronization

LL_CRS_GenerateEvent_SWSYNC

Function name	**__STATIC_INLINE void LL_CRS_GenerateEvent_SWSYNC (void)**
Function description	Generate software SYNC event.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR SWSYNC LL_CRS_GenerateEvent_SWSYNC

LL_CRS_GetFreqErrorDirection

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorDirection (void)**
Function description	Get the frequency error direction latched in the time of the last SYNC event.
Return values	• **Returned:** value can be one of the following values: – LL_CRS_FREQ_ERROR_DIR_UP – LL_CRS_FREQ_ERROR_DIR_DOWN
Reference Manual to LL API cross reference:	• ISR FEDIR LL_CRS_GetFreqErrorDirection

LL_CRS_GetFreqErrorCapture

Function name	**__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorCapture (void)**
Function description	Get the frequency error counter value latched in the time of the last SYNC event.
Return values	• **A:** number between Min_Data = 0x0000 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• ISR FECAP LL_CRS_GetFreqErrorCapture

LL_CRS_IsActiveFlag_SYNCOK

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCOK (void)**
Function description	Check if SYNC event OK signal occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SYNCOKF LL_CRS_IsActiveFlag_SYNCOK

LL_CRS_IsActiveFlag_SYNCWARN

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCWARN (void)**
Function description	Check if SYNC warning signal occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SYNCWARNF LL_CRS_IsActiveFlag_SYNCWARN

LL_CRS_IsActiveFlag_ERR

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ERR (void)**
Function description	Check if Synchronization or trimming error signal occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ERRF LL_CRS_IsActiveFlag_ERR

LL_CRS_IsActiveFlag_ESYNC

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ESYNC (void)**
Function description	Check if Expected SYNC signal occurred or not.
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• ISR ESYNCF LL_CRS_IsActiveFlag_ESYNC

LL_CRS_IsActiveFlag_SYNCERR

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCERR (void)**
Function description	Check if SYNC error signal occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SYNCERR LL_CRS_IsActiveFlag_SYNCERR

LL_CRS_IsActiveFlag_SYNCMISS

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCMISS (void)**
Function description	Check if SYNC missed error signal occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SYNCMISS LL_CRS_IsActiveFlag_SYNCMISS

LL_CRS_IsActiveFlag_TRIMOVF

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_TRIMOVF (void)**
Function description	Check if Trimming overflow or underflow occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TRIMOVF LL_CRS_IsActiveFlag_TRIMOVF

LL_CRS_ClearFlag_SYNCOK

Function name	**__STATIC_INLINE void LL_CRS_ClearFlag_SYNCOK (void)**
Function description	Clear the SYNC event OK flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR SYNCOKC LL_CRS_ClearFlag_SYNCOK

LL_CRS_ClearFlag_SYNCWARN

Function name	**__STATIC_INLINE void LL_CRS_ClearFlag_SYNCWARN (void)**

Function description	Clear the SYNC warning flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR SYNCWARNC LL_CRS_ClearFlag_SYNCWARN

LL_CRS_ClearFlag_ERR

Function name	**__STATIC_INLINE void LL_CRS_ClearFlag_ERR (void)**
Function description	Clear TRIMOVF, SYNCMISS and SYNCERR bits and consequently also the ERR flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR ERRC LL_CRS_ClearFlag_ERR

LL_CRS_ClearFlag_ESYNC

Function name	**__STATIC_INLINE void LL_CRS_ClearFlag_ESYNC (void)**
Function description	Clear Expected SYNC flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR ESYNCC LL_CRS_ClearFlag_ESYNC

LL_CRS_EnableIT_SYNCOK

Function name	**__STATIC_INLINE void LL_CRS_EnableIT_SYNCOK (void)**
Function description	Enable SYNC event OK interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR SYNCOKIE LL_CRS_EnableIT_SYNCOK

LL_CRS_DisableIT_SYNCOK

Function name	**__STATIC_INLINE void LL_CRS_DisableIT_SYNCOK (void)**
Function description	Disable SYNC event OK interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR SYNCOKIE LL_CRS_DisableIT_SYNCOK

LL_CRS_IsEnabledIT_SYNCOK

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCOK**

	(void)
Function description	Check if SYNC event OK interrupt is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR SYNCOKIE LL_CRS_IsEnabledIT_SYNCOK

LL_CRS_EnableIT_SYNCWARN

Function name	**__STATIC_INLINE void LL_CRS_EnableIT_SYNCWARN (void)**
Function description	Enable SYNC warning interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR SYNCWARNIE LL_CRS_EnableIT_SYNCWARN

LL_CRS_DisableIT_SYNCWARN

Function name	**__STATIC_INLINE void LL_CRS_DisableIT_SYNCWARN (void)**
Function description	Disable SYNC warning interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR SYNCWARNIE LL_CRS_DisableIT_SYNCWARN

LL_CRS_IsEnabledIT_SYNCWARN

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCWARN (void)**
Function description	Check if SYNC warning interrupt is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR SYNCWARNIE LL_CRS_IsEnabledIT_SYNCWARN

LL_CRS_EnableIT_ERR

Function name	**__STATIC_INLINE void LL_CRS_EnableIT_ERR (void)**
Function description	Enable Synchronization or trimming error interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR ERRIE LL_CRS_EnableIT_ERR

LL_CRS_DisableIT_ERR

Function name	**__STATIC_INLINE void LL_CRS_DisableIT_ERR (void)**
Function description	Disable Synchronization or trimming error interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR ERRIE LL_CRS_DisableIT_ERR

LL_CRS_IsEnabledIT_ERR

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ERR (void)**
Function description	Check if Synchronization or trimming error interrupt is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR ERRIE LL_CRS_IsEnabledIT_ERR

LL_CRS_EnableIT_ESYNC

Function name	**__STATIC_INLINE void LL_CRS_EnableIT_ESYNC (void)**
Function description	Enable Expected SYNC interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR ESYNCIE LL_CRS_EnableIT_ESYNC

LL_CRS_DisableIT_ESYNC

Function name	**__STATIC_INLINE void LL_CRS_DisableIT_ESYNC (void)**
Function description	Disable Expected SYNC interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR ESYNCIE LL_CRS_DisableIT_ESYNC

LL_CRS_IsEnabledIT_ESYNC

Function name	**__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ESYNC (void)**
Function description	Check if Expected SYNC interrupt is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR ESYNCIE LL_CRS_IsEnabledIT_ESYNC

LL_CRS_DeInit

Function name	**ErrorStatus LL_CRS_DeInit (void)**
Function description	De-Initializes CRS peripheral registers to their default reset values.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: CRS registers are de-initialized – ERROR: not applicable

78.2 CRS Firmware driver defines

78.2.1 CRS

Default Values

LL_CRS_RELOADVALUE_DEFAULT

Notes:

- The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB)

LL_CRS_ERRORLIMIT_DEFAULT

LL_CRS_HSI48CALIBRATION_DEFAULT

Notes:

- The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency

Frequency Error Direction

LL_CRS_FREQ_ERROR_DIR_UP	Upcounting direction, the actual frequency is above the target
LL_CRS_FREQ_ERROR_DIR_DOWN	Downcounting direction, the actual frequency is below the target

Get Flags Defines

LL_CRS_ISR_SYNCOKF

LL_CRS_ISR_SYNCWARNF

LL_CRS_ISR_ERRF

LL_CRS_ISR_ESYNCF

LL_CRS_ISR_SYNCERR

LL_CRS_ISR_SYNCMISS

LL_CRS_ISR_TRIMOVF

IT Defines

LL_CRS_CR_SYNCOKIE

LL_CRS_CR_SYNCWARNIE

LL_CRS_CR_ERRIE

LL_CRS_CR_ESYNCIE

Synchronization Signal Divider

LL_CRS_SYNC_DIV_1	Synchro Signal not divided (default)
LL_CRS_SYNC_DIV_2	Synchro Signal divided by 2
LL_CRS_SYNC_DIV_4	Synchro Signal divided by 4
LL_CRS_SYNC_DIV_8	Synchro Signal divided by 8
LL_CRS_SYNC_DIV_16	Synchro Signal divided by 16
LL_CRS_SYNC_DIV_32	Synchro Signal divided by 32
LL_CRS_SYNC_DIV_64	Synchro Signal divided by 64
LL_CRS_SYNC_DIV_128	Synchro Signal divided by 128

Synchronization Signal Polarity

LL_CRS_SYNC_POLARITY_RISING	Synchro Active on rising edge (default)
LL_CRS_SYNC_POLARITY_FALLING	Synchro Active on falling edge

Synchronization Signal Source

LL_CRS_SYNC_SOURCE_GPIO	Synchro Signal soucre GPIO
LL_CRS_SYNC_SOURCE_LSE	Synchro Signal source LSE
LL_CRS_SYNC_SOURCE_USB	Synchro Signal source USB SOF (default)

Exported_Macros_Calculate_Reload

__LL_CRS_CALC_CALCULATE_RELOADVALUE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- Reload: value (in Hz)

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

Common Write and read registers Macros

LL_CRS_WriteReg

Description:

- Write a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRS_ReadReg

Description:

- Read a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be read

Return value:

- Register: value

79 LL DAC Generic Driver

79.1 DAC Firmware driver registers structures

79.1.1 LL_DAC_InitTypeDef

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t WaveAutoGeneration***
- ***uint32_t WaveAutoGenerationConfig***
- ***uint32_t OutputBuffer***
- ***uint32_t OutputConnection***
- ***uint32_t OutputMode***

Field Documentation

- ***uint32_t LL_DAC_InitTypeDef::TriggerSource***
  Set the conversion trigger source for the selected DAC channel: internal (SW start) or from external IP (timer event, external interrupt line). This parameter can be a value of ***DAC_LL_EC_TRIGGER_SOURCE***This feature can be modified afterwards using unitary function **LL_DAC_SetTriggerSource()**.
- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGeneration***
  Set the waveform automatic generation mode for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_WAVE_AUTO_GENERATION_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetWaveAutoGeneration()**.
- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGenerationConfig***
  Set the waveform automatic generation mode for the selected DAC channel. If waveform automatic generation mode is set to noise, this parameter can be a value of ***DAC_LL_EC_WAVE_NOISE_LFSR_UNMASK_BITS*** If waveform automatic generation mode is set to triangle, this parameter can be a value of ***DAC_LL_EC_WAVE_TRIANGLE_AMPLITUDE***
  Note: If waveform automatic generation mode is disabled, this parameter is discarded. This feature can be modified afterwards using unitary function **LL_DAC_SetWaveNoiseLFSR()** or **LL_DAC_SetWaveTriangleAmplitude()**, depending on the wave automatic generation selected.
- ***uint32_t LL_DAC_InitTypeDef::OutputBuffer***
  Set the output buffer for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_BUFFER***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputBuffer()**.
- ***uint32_t LL_DAC_InitTypeDef::OutputConnection***
  Set the output connection for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_CONNECTION***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputConnection()**.
- ***uint32_t LL_DAC_InitTypeDef::OutputMode***
  Set the output mode normal or sample-and-hold for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputMode()**.

79.2 DAC Firmware driver API description

79.2.1 Detailed description of functions

LL_DAC_SetMode

Function name	**__STATIC_INLINE void LL_DAC_SetMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t ChannelMode)**
Function description	Set the operating mode for the selected DAC channel: calibration or normal operating mode.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. • **ChannelMode:** This parameter can be one of the following values: – LL_DAC_MODE_NORMAL_OPERATION – LL_DAC_MODE_CALIBRATION
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CEN1 LL_DAC_SetMode • CR CEN2 LL_DAC_SetMode

LL_DAC_GetMode

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the operating mode for the selected DAC channel: calibration or normal operating mode.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_MODE_NORMAL_OPERATION – LL_DAC_MODE_CALIBRATION
Reference Manual to LL API cross reference:	• CR CEN1 LL_DAC_GetMode • CR CEN2 LL_DAC_GetMode

LL_DAC_SetTrimmingValue

Function name	**__STATIC_INLINE void LL_DAC_SetTrimmingValue (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t**

	TrimmingValue)
Function description	Set the offset trimming value for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **TrimmingValue:** Value between Min_Data=0x00 and Max_Data=0x1F
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR OTRIM1 LL_DAC_SetTrimmingValue • CCR OTRIM2 LL_DAC_SetTrimmingValue

LL_DAC_GetTrimmingValue

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetTrimmingValue (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the offset trimming value for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **TrimmingValue:** Value between Min_Data=0x00 and Max_Data=0x1F
Reference Manual to LL API cross reference:	• CCR OTRIM1 LL_DAC_GetTrimmingValue • CCR OTRIM2 LL_DAC_GetTrimmingValue

LL_DAC_SetTriggerSource

Function name	**__STATIC_INLINE void LL_DAC_SetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriggerSource)**
Function description	Set the conversion trigger source for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **TriggerSource:** This parameter can be one of the following values:

- LL_DAC_TRIG_SOFTWARE
- LL_DAC_TRIG_EXT_TIM2_TRGO
- LL_DAC_TRIG_EXT_TIM4_TRGO
- LL_DAC_TRIG_EXT_TIM5_TRGO
- LL_DAC_TRIG_EXT_TIM6_TRGO
- LL_DAC_TRIG_EXT_TIM7_TRGO
- LL_DAC_TRIG_EXT_TIM8_TRGO
- LL_DAC_TRIG_EXT_EXTI_LINE9

Return values
- **None:**

Notes
- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- To set conversion trigger source, DAC channel must be disabled. Otherwise, the setting is discarded.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:
- CR TSEL1 LL_DAC_SetTriggerSource
- CR TSEL2 LL_DAC_SetTriggerSource

LL_DAC_GetTriggerSource

Function name
__STATIC_INLINE uint32_t LL_DAC_GetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Get the conversion trigger source for the selected DAC channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

Return values
- **Returned:** value can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_TIM5_TRGO
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM8_TRGO
  - LL_DAC_TRIGGER_EXT_IT9

Notes
- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:
- CR TSEL1 LL_DAC_GetTriggerSource
- CR TSEL2 LL_DAC_GetTriggerSource

LL_DAC_SetWaveAutoGeneration

Function name	**__STATIC_INLINE void LL_DAC_SetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t WaveAutoGeneration)**
Function description	Set the waveform automatic generation mode for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **WaveAutoGeneration:** This parameter can be one of the following values: – LL_DAC_WAVE_AUTO_GENERATION_NONE – LL_DAC_WAVE_AUTO_GENERATION_NOISE – LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR WAVE1 LL_DAC_SetWaveAutoGeneration • CR WAVE2 LL_DAC_SetWaveAutoGeneration

LL_DAC_GetWaveAutoGeneration

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the waveform automatic generation mode for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_WAVE_AUTO_GENERATION_NONE – LL_DAC_WAVE_AUTO_GENERATION_NOISE – LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE
Reference Manual to LL API cross reference:	• CR WAVE1 LL_DAC_GetWaveAutoGeneration • CR WAVE2 LL_DAC_GetWaveAutoGeneration

LL_DAC_SetWaveNoiseLFSR

Function name	**__STATIC_INLINE void LL_DAC_SetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t NoiseLFSRMask)**
Function description	Set the noise waveform generation for the selected DAC channel:

	Noise mode and parameters LFSR (linear feedback shift register).
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **NoiseLFSRMask:** This parameter can be one of the following values: – LL_DAC_NOISE_LFSR_UNMASK_BIT0 – LL_DAC_NOISE_LFSR_UNMASK_BITS1_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS2_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS3_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS4_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS5_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS6_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS7_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS8_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS9_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS10_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS11_0
Return values	• **None:**
Notes	• For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration(). • This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).
Reference Manual to LL API cross reference:	• CR MAMP1 LL_DAC_SetWaveNoiseLFSR • CR MAMP2 LL_DAC_SetWaveNoiseLFSR

LL_DAC_GetWaveNoiseLFSR

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Set the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_NOISE_LFSR_UNMASK_BIT0 – LL_DAC_NOISE_LFSR_UNMASK_BITS1_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS2_0 – LL_DAC_NOISE_LFSR_UNMASK_BITS3_0

- LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
- LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
- LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
- LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
- LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
- LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
- LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
- LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Reference Manual to LL API cross reference:
- CR MAMP1 LL_DAC_GetWaveNoiseLFSR
- CR MAMP2 LL_DAC_GetWaveNoiseLFSR

LL_DAC_SetWaveTriangleAmplitude

Function name
: **__STATIC_INLINE void LL_DAC_SetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriangleAmplitude)**

Function description
: Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)
- **TriangleAmplitude:** This parameter can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Return values
- **None:**

Notes
- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:
- CR MAMP1 LL_DAC_SetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_SetWaveTriangleAmplitude

LL_DAC_GetWaveTriangleAmplitude

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_TRIANGLE_AMPLITUDE_1 – LL_DAC_TRIANGLE_AMPLITUDE_3 – LL_DAC_TRIANGLE_AMPLITUDE_7 – LL_DAC_TRIANGLE_AMPLITUDE_15 – LL_DAC_TRIANGLE_AMPLITUDE_31 – LL_DAC_TRIANGLE_AMPLITUDE_63 – LL_DAC_TRIANGLE_AMPLITUDE_127 – LL_DAC_TRIANGLE_AMPLITUDE_255 – LL_DAC_TRIANGLE_AMPLITUDE_511 – LL_DAC_TRIANGLE_AMPLITUDE_1023 – LL_DAC_TRIANGLE_AMPLITUDE_2047 – LL_DAC_TRIANGLE_AMPLITUDE_4095
Reference Manual to LL API cross reference:	• CR MAMP1 LL_DAC_GetWaveTriangleAmplitude • CR MAMP2 LL_DAC_GetWaveTriangleAmplitude

LL_DAC_ConfigOutput

Function name	**__STATIC_INLINE void LL_DAC_ConfigOutput (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputMode, uint32_t OutputBuffer, uint32_t OutputConnection)**
Function description	Set the output for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **OutputMode:** This parameter can be one of the following values: – LL_DAC_OUTPUT_MODE_NORMAL – LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD • **OutputBuffer:** This parameter can be one of the following values: – LL_DAC_OUTPUT_BUFFER_ENABLE – LL_DAC_OUTPUT_BUFFER_DISABLE

	• **OutputConnection:** This parameter can be one of the following values: – LL_DAC_OUTPUT_CONNECT_GPIO – LL_DAC_OUTPUT_CONNECT_INTERNAL
Return values	• **None:**
Notes	• This function set several features: mode normal or sample-and-holdbufferconnection to GPIO or internal path. These features can also be set individually using dedicated functions:LL_DAC_SetOutputBuffer()LL_DAC_SetOutputMode()LL_DAC_SetOutputConnection() • On this STM32 serie, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path). • Mode sample-and-hold requires an external capacitor to be connected between DAC channel output and ground. Capacitor value depends on load on DAC channel output and sample-and-hold timings configured. As indication, capacitor typical value is 100nF (refer to device datasheet, parameter "CSH").
Reference Manual to LL API cross reference:	• CR MODE1 LL_DAC_ConfigOutput • CR MODE2 LL_DAC_ConfigOutput

LL_DAC_SetOutputMode

Function name	**__STATIC_INLINE void LL_DAC_SetOutputMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputMode)**
Function description	Set the output mode normal or sample-and-hold for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **OutputMode:** This parameter can be one of the following values: – LL_DAC_OUTPUT_MODE_NORMAL – LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD
Return values	• **None:**
Notes	• Mode sample-and-hold requires an external capacitor to be connected between DAC channel output and ground.

	Capacitor value depends on load on DAC channel output and sample-and-hold timings configured. As indication, capacitor typical value is 100nF (refer to device datasheet, parameter "CSH").
Reference Manual to LL API cross reference:	• CR MODE1 LL_DAC_SetOutputMode • CR MODE2 LL_DAC_SetOutputMode

LL_DAC_GetOutputMode

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetOutputMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the output mode normal or sample-and-hold for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_OUTPUT_MODE_NORMAL – LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD
Reference Manual to LL API cross reference:	• CR MODE1 LL_DAC_GetOutputMode • CR MODE2 LL_DAC_GetOutputMode

LL_DAC_SetOutputBuffer

Function name	**__STATIC_INLINE void LL_DAC_SetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputBuffer)**
Function description	Set the output buffer for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **OutputBuffer:** This parameter can be one of the following values: – LL_DAC_OUTPUT_BUFFER_ENABLE – LL_DAC_OUTPUT_BUFFER_DISABLE
Return values	• **None:**
Notes	• On this STM32 serie, when buffer is enabled, its offset can be trimmed: factory calibration default values can be replaced by user trimming values, using function

LL_DAC_SetTrimmingValue().

Reference Manual to LL API cross reference:	• CR MODE1 LL_DAC_SetOutputBuffer • CR MODE2 LL_DAC_SetOutputBuffer

LL_DAC_GetOutputBuffer

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the output buffer state for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_OUTPUT_BUFFER_ENABLE – LL_DAC_OUTPUT_BUFFER_DISABLE
Reference Manual to LL API cross reference:	• CR MODE1 LL_DAC_GetOutputBuffer • CR MODE2 LL_DAC_GetOutputBuffer

LL_DAC_SetOutputConnection

Function name	**__STATIC_INLINE void LL_DAC_SetOutputConnection (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputConnection)**
Function description	Set the output connection for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **OutputConnection:** This parameter can be one of the following values: – LL_DAC_OUTPUT_CONNECT_GPIO – LL_DAC_OUTPUT_CONNECT_INTERNAL
Return values	• **None:**
Notes	• On this STM32 serie, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output

	connection is also connected to internal path (both connections to GPIO pin and internal path).
Reference Manual to LL API cross reference:	• CR MODE1 LL_DAC_SetOutputConnection • CR MODE2 LL_DAC_SetOutputConnection

LL_DAC_GetOutputConnection

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetOutputConnection (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the output connection for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Returned:** value can be one of the following values: – LL_DAC_OUTPUT_CONNECT_GPIO – LL_DAC_OUTPUT_CONNECT_INTERNAL
Notes	• On this STM32 serie, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).
Reference Manual to LL API cross reference:	• CR MODE1 LL_DAC_GetOutputConnection • CR MODE2 LL_DAC_GetOutputConnection

LL_DAC_SetSampleAndHoldSampleTime

Function name	**__STATIC_INLINE void LL_DAC_SetSampleAndHoldSampleTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t SampleTime)**
Function description	Set the sample-and-hold timing for the selected DAC channel: sample time.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **SampleTime:** Value between Min_Data=0x000 and Max_Data=0x3FF

Return values	• **None:**
Notes	• Sample time must be set when DAC channel is disabled or during DAC operation when DAC channel flag BWSTx is reset, otherwise the setting is ignored. Check BWSTx flag state using function "LL_DAC_IsActiveFlag_BWSTx()".
Reference Manual to LL API cross reference:	• SHSR1 TSAMPLE1 LL_DAC_SetSampleAndHoldSampleTime • SHSR2 TSAMPLE2 LL_DAC_SetSampleAndHoldSampleTime

LL_DAC_GetSampleAndHoldSampleTime

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldSampleTime (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the sample-and-hold timing for the selected DAC channel: sample time.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0x3FF
Reference Manual to LL API cross reference:	• SHSR1 TSAMPLE1 LL_DAC_GetSampleAndHoldSampleTime • SHSR2 TSAMPLE2 LL_DAC_GetSampleAndHoldSampleTime

LL_DAC_SetSampleAndHoldHoldTime

Function name	**__STATIC_INLINE void LL_DAC_SetSampleAndHoldHoldTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t HoldTime)**
Function description	Set the sample-and-hold timing for the selected DAC channel: hold time.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **HoldTime:** Value between Min_Data=0x000 and Max_Data=0x3FF
Return values	• **None:**
Reference Manual to LL API cross	• SHHR THOLD1 LL_DAC_SetSampleAndHoldHoldTime

reference:	• SHHR THOLD2 LL_DAC_SetSampleAndHoldHoldTime

LL_DAC_GetSampleAndHoldHoldTime

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldHoldTime (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the sample-and-hold timing for the selected DAC channel: hold time.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0x3FF
Reference Manual to LL API cross reference:	• SHHR THOLD1 LL_DAC_GetSampleAndHoldHoldTime • SHHR THOLD2 LL_DAC_GetSampleAndHoldHoldTime

LL_DAC_SetSampleAndHoldRefreshTime

Function name	**__STATIC_INLINE void LL_DAC_SetSampleAndHoldRefreshTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t RefreshTime)**
Function description	Set the sample-and-hold timing for the selected DAC channel: refresh time.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **RefreshTime:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• SHRR TREFRESH1 LL_DAC_SetSampleAndHoldRefreshTime • SHRR TREFRESH2 LL_DAC_SetSampleAndHoldRefreshTime

LL_DAC_GetSampleAndHoldRefreshTime

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldRefreshTime (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the sample-and-hold timing for the selected DAC channel:

refresh time.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2 (1)

Return values
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:
- SHRR TREFRESH1 LL_DAC_GetSampleAndHoldRefreshTime
- SHRR TREFRESH2 LL_DAC_GetSampleAndHoldRefreshTime

LL_DAC_SetWaveMode

Function name
__STATIC_INLINE void LL_DAC_SetWaveMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t WaveMode)

Function description

LL_DAC_GetWaveMode

Function name
__STATIC_INLINE uint32_t LL_DAC_GetWaveMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

LL_DAC_EnableDMAReq

Function name
__STATIC_INLINE void LL_DAC_EnableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Enable DAC DMA transfer request of the selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2 (1)

Return values
- **None:**

Notes
- To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:
- CR DMAEN1 LL_DAC_EnableDMAReq
- CR DMAEN2 LL_DAC_EnableDMAReq

LL_DAC_DisableDMAReq

Function name	**__STATIC_INLINE void LL_DAC_DisableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Disable DAC DMA transfer request of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **None:**
Notes	• To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().
Reference Manual to LL API cross reference:	• CR DMAEN1 LL_DAC_DisableDMAReq • CR DMAEN2 LL_DAC_DisableDMAReq

LL_DAC_IsDMAReqEnabled

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsDMAReqEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get DAC DMA transfer request state of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR DMAEN1 LL_DAC_IsDMAReqEnabled • CR DMAEN2 LL_DAC_IsDMAReqEnabled

LL_DAC_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_DAC_DMA_GetRegAddr (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Register)**
Function description	Function to help to configure DMA transfer to DAC: retrieve the DAC register address from DAC instance and a list of DAC registers intended to be used (most commonly) with DMA transfer.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)

- **Register:** This parameter can be one of the following values:
  - LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED
  - LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED
  - LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

Return values

- **DAC:** register address

Notes

- These DAC registers are data holding registers: when DAC conversion is requested, DAC generates a DMA transfer request to have data available in DAC data holding registers.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, (uint32_t)&< array or variable >, LL_DAC_DMA_GetRegAddr(DAC1, LL_DAC_CHANNEL_1, LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED), LL_DMA_DIRECTION_MEMORY_TO_PERIPH);

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12L1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR8R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12R2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR12L2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR8R2 DACC2DHR LL_DAC_DMA_GetRegAddr

LL_DAC_Enable

Function name

__STATIC_INLINE void LL_DAC_Enable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

Return values

- **None:**

Notes

- After enable from off state, DAC channel requires a delay for output voltage to reach accuracy +/- 1 LSB. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_Enable
- CR EN2 LL_DAC_Enable

LL_DAC_Disable

Function name

__STATIC_INLINE void LL_DAC_Disable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following

values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2 (1)

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR EN1 LL_DAC_Disable
- CR EN2 LL_DAC_Disable

LL_DAC_IsEnabled

Function name
__STATIC_INLINE uint32_t LL_DAC_IsEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Get DAC enable state of the selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2 (1)

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CR EN1 LL_DAC_IsEnabled
- CR EN2 LL_DAC_IsEnabled

LL_DAC_EnableTrigger

Function name
__STATIC_INLINE void LL_DAC_EnableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Enable DAC trigger of the selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2 (1)

Return values
- **None:**

Notes
- - If DAC trigger is disabled, DAC conversion is performed automatically once the data holding register is updated, using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ... If DAC trigger is enabled, DAC conversion is performed only when a hardware of software trigger event is occurring. Select trigger source using function LL_DAC_SetTriggerSource().

Reference Manual to LL API cross
- CR TEN1 LL_DAC_EnableTrigger

reference:	• CR TEN2 LL_DAC_EnableTrigger

LL_DAC_DisableTrigger

Function name	**__STATIC_INLINE void LL_DAC_DisableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Disable DAC trigger of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR TEN1 LL_DAC_DisableTrigger • CR TEN2 LL_DAC_DisableTrigger

LL_DAC_IsTriggerEnabled

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsTriggerEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get DAC trigger state of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR TEN1 LL_DAC_IsTriggerEnabled • CR TEN2 LL_DAC_IsTriggerEnabled

LL_DAC_TrigSWConversion

Function name	**__STATIC_INLINE void LL_DAC_TrigSWConversion (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Trig DAC conversion by software for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can a combination of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)

Return values	• **None:**
Notes	• Preliminarily, DAC trigger must be set to software trigger using function LL_DAC_SetTriggerSource() with parameter "LL_DAC_TRIGGER_SOFTWARE". and DAC trigger must be enabled using function LL_DAC_EnableTrigger(). • For devices featuring DAC with 2 channels: this function can perform a SW start of both DAC channels simultaneously. Two channels can be selected as parameter. Example: (LL_DAC_CHANNEL_1 | LL_DAC_CHANNEL_2)
Reference Manual to LL API cross reference:	• SWTRIGR SWTRIG1 LL_DAC_TrigSWConversion • SWTRIGR SWTRIG2 LL_DAC_TrigSWConversion

LL_DAC_ConvertData12RightAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertData12RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)**
Function description	Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DHR12R1 DACC1DHR LL_DAC_ConvertData12RightAligned • DHR12R2 DACC2DHR LL_DAC_ConvertData12RightAligned

LL_DAC_ConvertData12LeftAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertData12LeftAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)**
Function description	Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **Data:** Value between Min_Data=0x000 and

Max_Data=0xFFF

Return values	• **None:**
Reference Manual to LL API cross reference:	• DHR12L1 DACC1DHR LL_DAC_ConvertData12LeftAligned • DHR12L2 DACC2DHR LL_DAC_ConvertData12LeftAligned

LL_DAC_ConvertData8RightAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertData8RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)**
Function description	Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **Data:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DHR8R1 DACC1DHR LL_DAC_ConvertData8RightAligned • DHR8R2 DACC2DHR LL_DAC_ConvertData8RightAligned

LL_DAC_ConvertDualData12RightAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertDualData12RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)**
Function description	Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for both DAC channels.
Parameters	• **DACx:** DAC instance • **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF • **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DHR12RD DACC1DHR LL_DAC_ConvertDualData12RightAligned • DHR12RD DACC2DHR LL_DAC_ConvertDualData12RightAligned

LL_DAC_ConvertDualData12LeftAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertDualData12LeftAligned (DAC_TypeDef ***

	DACx, uint32_t DataChannel1, uint32_t DataChannel2)
Function description	Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for both DAC channels.
Parameters	• **DACx:** DAC instance • **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF • **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DHR12LD DACC1DHR LL_DAC_ConvertDualData12LeftAligned • DHR12LD DACC2DHR LL_DAC_ConvertDualData12LeftAligned

LL_DAC_ConvertDualData8RightAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertDualData8RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)**
Function description	Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for both DAC channels.
Parameters	• **DACx:** DAC instance • **DataChannel1:** Value between Min_Data=0x00 and Max_Data=0xFF • **DataChannel2:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DHR8RD DACC1DHR LL_DAC_ConvertDualData8RightAligned • DHR8RD DACC2DHR LL_DAC_ConvertDualData8RightAligned

LL_DAC_RetrieveOutputData

Function name	**__STATIC_INLINE uint32_t LL_DAC_RetrieveOutputData (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Retrieve output data currently generated for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1)
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0xFFF
Notes	• Whatever alignment and resolution settings (using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()":

	LL_DAC_ConvertData12RightAligned(), ...), output data format is 12 bits right aligned (LSB aligned on bit 0).
Reference Manual to LL API cross reference:	• DOR1 DACC1DOR LL_DAC_RetrieveOutputData • DOR2 DACC2DOR LL_DAC_RetrieveOutputData

LL_DAC_IsActiveFlag_CAL1

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_CAL1 (DAC_TypeDef * DACx)**
Function description	Get DAC calibration offset flag for DAC channel 1.
Parameters	• **DACx:** DAC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CAL_FLAG1 LL_DAC_IsActiveFlag_CAL1

LL_DAC_IsActiveFlag_CAL2

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_CAL2 (DAC_TypeDef * DACx)**
Function description	Get DAC calibration offset flag for DAC channel 2.
Parameters	• **DACx:** DAC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CAL_FLAG2 LL_DAC_IsActiveFlag_CAL2

LL_DAC_IsActiveFlag_BWST1

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_BWST1 (DAC_TypeDef * DACx)**
Function description	Get DAC busy writing sample time flag for DAC channel 1.
Parameters	• **DACx:** DAC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR BWST1 LL_DAC_IsActiveFlag_BWST1

LL_DAC_IsActiveFlag_BWST2

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_BWST2 (DAC_TypeDef * DACx)**
Function description	Get DAC busy writing sample time flag for DAC channel 2.
Parameters	• **DACx:** DAC instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR BWST2 LL_DAC_IsActiveFlag_BWST2

LL_DAC_IsActiveFlag_DMAUDR1

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR1 (DAC_TypeDef * DACx)**
Function description	Get DAC underrun flag for DAC channel 1.
Parameters	• **DACx:** DAC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR DMAUDR1 LL_DAC_IsActiveFlag_DMAUDR1

LL_DAC_IsActiveFlag_DMAUDR2

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR2 (DAC_TypeDef * DACx)**
Function description	Get DAC underrun flag for DAC channel 2.
Parameters	• **DACx:** DAC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR DMAUDR2 LL_DAC_IsActiveFlag_DMAUDR2

LL_DAC_ClearFlag_DMAUDR1

Function name	**__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR1 (DAC_TypeDef * DACx)**
Function description	Clear DAC underrun flag for DAC channel 1.
Parameters	• **DACx:** DAC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR DMAUDR1 LL_DAC_ClearFlag_DMAUDR1

LL_DAC_ClearFlag_DMAUDR2

Function name	**__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR2 (DAC_TypeDef * DACx)**
Function description	Clear DAC underrun flag for DAC channel 2.
Parameters	• **DACx:** DAC instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• SR DMAUDR2 LL_DAC_ClearFlag_DMAUDR2

LL_DAC_EnableIT_DMAUDR1

Function name	**__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR1 (DAC_TypeDef * DACx)**
Function description	Enable DMA underrun interrupt for DAC channel 1.
Parameters	• **DACx:** DAC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR DMAUDRIE1 LL_DAC_EnableIT_DMAUDR1

LL_DAC_EnableIT_DMAUDR2

Function name	**__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR2 (DAC_TypeDef * DACx)**
Function description	Enable DMA underrun interrupt for DAC channel 2.
Parameters	• **DACx:** DAC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR DMAUDRIE2 LL_DAC_EnableIT_DMAUDR2

LL_DAC_DisableIT_DMAUDR1

Function name	**__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR1 (DAC_TypeDef * DACx)**
Function description	Disable DMA underrun interrupt for DAC channel 1.
Parameters	• **DACx:** DAC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR DMAUDRIE1 LL_DAC_DisableIT_DMAUDR1

LL_DAC_DisableIT_DMAUDR2

Function name	**__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR2 (DAC_TypeDef * DACx)**
Function description	Disable DMA underrun interrupt for DAC channel 2.
Parameters	• **DACx:** DAC instance
Return values	• **None:**
Reference Manual to LL API cross	• CR DMAUDRIE2 LL_DAC_DisableIT_DMAUDR2

reference:

LL_DAC_IsEnabledIT_DMAUDR1

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR1 (DAC_TypeDef * DACx)**
Function description	Get DMA underrun interrupt for DAC channel 1.
Parameters	• **DACx:** DAC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR DMAUDRIE1 LL_DAC_IsEnabledIT_DMAUDR1

LL_DAC_IsEnabledIT_DMAUDR2

Function name	**__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR2 (DAC_TypeDef * DACx)**
Function description	Get DMA underrun interrupt for DAC channel 2.
Parameters	• **DACx:** DAC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR DMAUDRIE2 LL_DAC_IsEnabledIT_DMAUDR2

LL_DAC_DeInit

Function name	**ErrorStatus LL_DAC_DeInit (DAC_TypeDef * DACx)**
Function description	De-initialize registers of the selected DAC instance to their default reset values.
Parameters	• **DACx:** DAC instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: DAC registers are de-initialized – ERROR: not applicable

LL_DAC_Init

Function name	**ErrorStatus LL_DAC_Init (DAC_TypeDef * DACx, uint32_t DAC_Channel, LL_DAC_InitTypeDef * DAC_InitStruct)**
Function description	Initialize some features of DAC instance.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 (1) • **DAC_InitStruct:** Pointer to a LL_DAC_InitTypeDef structure

Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: DAC registers are initialized – ERROR: DAC registers are not initialized
Notes	• The setting of these parameters by function LL_DAC_Init() is conditioned to DAC state: DAC instance must be disabled.

LL_DAC_StructInit

Function name	**void LL_DAC_StructInit (LL_DAC_InitTypeDef * DAC_InitStruct)**
Function description	Set each LL_DAC_InitTypeDef field to default value.
Parameters	• **DAC_InitStruct:** pointer to a LL_DAC_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

79.3 DAC Firmware driver defines

79.3.1 DAC

DAC channels

LL_DAC_CHANNEL_1	DAC channel 1
LL_DAC_CHANNEL_2	DAC channel 2

DAC flags

LL_DAC_FLAG_DMAUDR1	DAC channel 1 flag DMA underrun
LL_DAC_FLAG_CAL1	DAC channel 1 flag offset calibration status
LL_DAC_FLAG_BWST1	DAC channel 1 flag busy writing sample time
LL_DAC_FLAG_DMAUDR2	DAC channel 2 flag DMA underrun
LL_DAC_FLAG_CAL2	DAC channel 2 flag offset calibration status
LL_DAC_FLAG_BWST2	DAC channel 2 flag busy writing sample time

Definitions of DAC hardware constraints delays

LL_DAC_DELAY_STARTUP_VOLTAGE_SETTLING_US	Delay for DAC channel voltage settling time from DAC channel startup (transition from disable to enable)
LL_DAC_DELAY_VOLTAGE_SETTLING_US	Delay for DAC channel voltage settling time

DAC interruptions

LL_DAC_IT_DMAUDRIE1	DAC channel 1 interruption DMA underrun
LL_DAC_IT_DMAUDRIE2	DAC channel 2 interruption DMA underrun

DAC literals legacy naming

LL_DAC_TRIGGER_SOFTWARE

LL_DAC_TRIGGER_TIM2_TRGO

LL_DAC_TRIGGER_TIM4_TRGO

LL_DAC_TRIGGER_TIM5_TRGO

LL_DAC_TRIGGER_TIM6_TRGO

LL_DAC_TRIGGER_TIM7_TRGO

LL_DAC_TRIGGER_TIM8_TRGO

LL_DAC_TRIGGER_EXT_IT9

LL_DAC_WAVEGENERATION_NONE

LL_DAC_WAVEGENERATION_NOISE

LL_DAC_WAVEGENERATION_TRIANGLE

LL_DAC_CONNECT_GPIO

LL_DAC_CONNECT_INTERNAL

DAC operating mode

LL_DAC_MODE_NORMAL_OPERATION	DAC channel in mode normal operation
LL_DAC_MODE_CALIBRATION	DAC channel in mode calibration

DAC channel output buffer

LL_DAC_OUTPUT_BUFFER_ENABLE	The selected DAC channel output is buffered: higher drive current capability, but also higher current consumption
LL_DAC_OUTPUT_BUFFER_DISABLE	The selected DAC channel output is not buffered: lower drive current capability, but also lower current consumption

DAC channel output connection

LL_DAC_OUTPUT_CONNECT_GPIO	The selected DAC channel output is connected to external pin
LL_DAC_OUTPUT_CONNECT_INTERNAL	The selected DAC channel output is connected to on-chip peripherals via internal paths. On this STM32 serie, output connection depends on output mode (normal or sample and hold) and output buffer state. Refer to comments of function

DAC channel output mode

LL_DAC_OUTPUT_MODE_NORMAL	The selected DAC channel output is on mode normal.
LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD	The selected DAC channel output is on mode sample-and-hold. Mode sample-and-hold requires an external capacitor, refer to description of function

DAC registers compliant with specific purpose

LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED	DAC channel data holding register 12 bits right aligned
LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED	DAC channel data holding

	register 12 bits left aligned
LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED	DAC channel data holding register 8 bits right aligned

DAC channel output resolution

LL_DAC_RESOLUTION_12B	DAC channel resolution 12 bits
LL_DAC_RESOLUTION_8B	DAC channel resolution 8 bits

DAC trigger source

LL_DAC_TRIG_SOFTWARE	DAC channel conversion trigger internal (SW start)
LL_DAC_TRIG_EXT_TIM2_TRGO	DAC channel conversion trigger from external IP: TIM2 TRGO.
LL_DAC_TRIG_EXT_TIM4_TRGO	DAC channel conversion trigger from external IP: TIM4 TRGO.
LL_DAC_TRIG_EXT_TIM5_TRGO	DAC channel conversion trigger from external IP: TIM5 TRGO.
LL_DAC_TRIG_EXT_TIM6_TRGO	DAC channel conversion trigger from external IP: TIM6 TRGO.
LL_DAC_TRIG_EXT_TIM7_TRGO	DAC channel conversion trigger from external IP: TIM7 TRGO.
LL_DAC_TRIG_EXT_TIM8_TRGO	DAC channel conversion trigger from external IP: TIM8 TRGO.
LL_DAC_TRIG_EXT_EXTI_LINE9	DAC channel conversion trigger from external IP: external interrupt line 9.

DAC waveform automatic generation mode

LL_DAC_WAVE_AUTO_GENERATION_NONE	DAC channel wave auto generation mode disabled.
LL_DAC_WAVE_AUTO_GENERATION_NOISE	DAC channel wave auto generation mode enabled, set generated noise waveform.
LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE	DAC channel wave auto generation mode enabled, set generated triangle waveform.

DAC wave generation - Noise LFSR unmask bits

LL_DAC_NOISE_LFSR_UNMASK_BIT0	Noise wave generation, unmask LFSR bit0, for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS1_0	Noise wave generation, unmask LFSR bits[1:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS2_0	Noise wave generation, unmask LFSR bits[2:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS3_0	Noise wave generation, unmask LFSR bits[3:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS4_0	Noise wave generation, unmask LFSR bits[4:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS5_0	Noise wave generation, unmask LFSR

	bits[5:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS6_0	Noise wave generation, unmask LFSR bits[6:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS7_0	Noise wave generation, unmask LFSR bits[7:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS8_0	Noise wave generation, unmask LFSR bits[8:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS9_0	Noise wave generation, unmask LFSR bits[9:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS10_0	Noise wave generation, unmask LFSR bits[10:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS11_0	Noise wave generation, unmask LFSR bits[11:0], for the selected DAC channel

DAC wave generation - Triangle amplitude

LL_DAC_TRIANGLE_AMPLITUDE_1	Triangle wave generation, amplitude of 1 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_3	Triangle wave generation, amplitude of 3 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_7	Triangle wave generation, amplitude of 7 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_15	Triangle wave generation, amplitude of 15 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_31	Triangle wave generation, amplitude of 31 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_63	Triangle wave generation, amplitude of 63 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_127	Triangle wave generation, amplitude of 127 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_255	Triangle wave generation, amplitude of 255 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_511	Triangle wave generation, amplitude of 512 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_1023	Triangle wave generation, amplitude of 1023 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_2047	Triangle wave generation, amplitude of 2047 LSB of DAC output range, for the selected

	DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_4095	Triangle wave generation, amplitude of 4095 LSB of DAC output range, for the selected DAC channel

DAC helper macro

__LL_DAC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get DAC channel number in decimal format from literals LL_DAC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

Return value:

- 1...2: (value "2" depending on DAC channel 2 availability)

Notes:

- The input can be a value from functions where a channel number is returned.

__LL_DAC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get DAC channel in literal format LL_DAC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: 1...2 (value "2" depending on DAC channel 2 availability)

Return value:

- Returned: value can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

Notes:

- If the input parameter does not correspond to a DAC channel, this macro returns value '0'.

__LL_DAC_DIGITAL_SCALE

Description:

- Helper macro to define the DAC conversion data full-scale digital value corresponding to the selected DAC resolution.

Parameters:

- __DAC_RESOLUTION__: This parameter

can be one of the following values:
- LL_DAC_RESOLUTION_12B
- LL_DAC_RESOLUTION_8B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- DAC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_DAC_CALC_VOLTAGE_TO_DATA

Description:

- Helper macro to calculate the DAC conversion data (unit: digital value) corresponding to a voltage (unit: mVolt).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __DAC_VOLTAGE__: Voltage to be generated by DAC channel (unit: mVolt).
- __DAC_RESOLUTION__: This parameter can be one of the following values:
  - LL_DAC_RESOLUTION_12B
  - LL_DAC_RESOLUTION_8B

Return value:

- DAC: conversion data (unit: digital value)

Notes:

- This helper macro is intended to provide input data in voltage rather than digital value, to be used with LL DAC functions such as LL_DAC_ConvertData12RightAligned(). Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

Common write and read registers macros

LL_DAC_WriteReg

Description:

- Write a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be written

- __VALUE__: Value to be written in the register

Return value:

- None

LL_DAC_ReadReg

Description:

- Read a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be read

Return value:

- Register: value

80 LL DMA2D Generic Driver

80.1 DMA2D Firmware driver registers structures

80.1.1 LL_DMA2D_InitTypeDef

Data Fields

- ***uint32_t Mode***
- ***uint32_t ColorMode***
- ***uint32_t OutputBlue***
- ***uint32_t OutputGreen***
- ***uint32_t OutputRed***
- ***uint32_t OutputAlpha***
- ***uint32_t OutputMemoryAddress***
- ***uint32_t OutputSwapMode***
- ***uint32_t LineOffsetMode***
- ***uint32_t LineOffset***
- ***uint32_t NbrOfLines***
- ***uint32_t NbrOfPixelsPerLines***
- ***uint32_t AlphaInversionMode***
- ***uint32_t RBSwapMode***

Field Documentation

- ***uint32_t LL_DMA2D_InitTypeDef::Mode***
  Specifies the DMA2D transfer mode.This parameter can be one value of ***DMA2D_LL_EC_MODE***. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::ColorMode***
  Specifies the color format of the output image.This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_COLOR_MODE***. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColorMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputBlue***
  Specifies the Blue value of the output image.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputGreen***
  Specifies the Green value of the output image.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x3F if RGB565 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.This parameter must be a number between

Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.

- ***uint32_t LL_DMA2D_InitTypeDef::OutputRed***
  Specifies the Red value of the output image.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputAlpha***
  Specifies the Alpha channel of the output image.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x01 if ARGB1555 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.This parameter is not considered if RGB888 or RGB565 color mode is selected. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputMemoryAddress***
  Specifies the memory address.This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputMemAddr()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputSwapMode***
  Specifies the output swap mode color format of the output image.This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_SWAP_MODE***. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputSwapMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::LineOffsetMode***
  Specifies the output line offset mode.This parameter can be one value of ***DMA2D_LL_EC_LINE_OFFSET_MODE***. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetLineOffsetMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::LineOffset***
  Specifies the output line offset value.This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF on STM32L496xx/STM32L4A6xx else between Min_Data = 0x0000 and Max_Data = 0xFFFF on other devices. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetLineOffset()**.
- ***uint32_t LL_DMA2D_InitTypeDef::NbrOfLines***
  Specifies the number of lines of the area to be transferred.This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetNbrOfLines()**.
- ***uint32_t LL_DMA2D_InitTypeDef::NbrOfPixelsPerLines***
  Specifies the number of pixels per lines of the area to be transfered.This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetNbrOfPixelsPerLines()**.
- ***uint32_t LL_DMA2D_InitTypeDef::AlphaInversionMode***
  Specifies the output alpha inversion mode.This parameter can be one value of

DMA2D_LL_EC_ALPHA_INVERSION. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputAlphaInvMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::RBSwapMode***
  Specifies the output Red Blue swap mode.This parameter can be one value of *DMA2D_LL_EC_RED_BLUE_SWAP*. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputRBSwapMode()**.

80.1.2 LL_DMA2D_LayerCfgTypeDef

Data Fields

- ***uint32_t MemoryAddress***
- ***uint32_t LineOffset***
- ***uint32_t ColorMode***
- ***uint32_t CLUTColorMode***
- ***uint32_t CLUTSize***
- ***uint32_t AlphaMode***
- ***uint32_t Alpha***
- ***uint32_t Blue***
- ***uint32_t Green***
- ***uint32_t Red***
- ***uint32_t CLUTMemoryAddress***
- ***uint32_t AlphaInversionMode***
- ***uint32_t RBSwapMode***

Field Documentation

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::MemoryAddress***
  Specifies the foreground or background memory address.This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetMemAddr()** for foreground layer,**LL_DMA2D_BGND_SetMemAddr()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::LineOffset***
  Specifies the foreground or background line offset value.This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetLineOffset()** for foreground layer,**LL_DMA2D_BGND_SetLineOffset()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::ColorMode***
  Specifies the foreground or background color mode.This parameter can be one value of *DMA2D_LL_EC_INPUT_COLOR_MODE*. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetColorMode()** for foreground layer,**LL_DMA2D_BGND_SetColorMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTColorMode***
  Specifies the foreground or background CLUT color mode.This parameter can be one value of *DMA2D_LL_EC_CLUT_COLOR_MODE*. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetCLUTColorMode()** for foreground layer,**LL_DMA2D_BGND_SetCLUTColorMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTSize***
  Specifies the foreground or background CLUT size.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetCLUTSize()** for foreground layer,**LL_DMA2D_BGND_SetCLUTSize()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::AlphaMode***
  Specifies the foreground or background alpha mode.This parameter can be one value of *DMA2D_LL_EC_ALPHA_MODE*. This parameter can be modified afterwards using

unitary functions **LL_DMA2D_FGND_SetAlphaMode()** for foreground layer,**LL_DMA2D_BGND_SetAlphaMode()** for background layer.

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Alpha***
  Specifies the foreground or background Alpha value.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetAlpha()** for foreground layer,**LL_DMA2D_BGND_SetAlpha()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Blue***
  Specifies the foreground or background Blue color value.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetBlueColor()** for foreground layer,**LL_DMA2D_BGND_SetBlueColor()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Green***
  Specifies the foreground or background Green color value.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetGreenColor()** for foreground layer,**LL_DMA2D_BGND_SetGreenColor()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Red***
  Specifies the foreground or background Red color value.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetRedColor()** for foreground layer,**LL_DMA2D_BGND_SetRedColor()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTMemoryAddress***
  Specifies the foreground or background CLUT memory address.This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetCLUTMemAddr()** for foreground layer,**LL_DMA2D_BGND_SetCLUTMemAddr()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::AlphaInversionMode***
  Specifies the foreground or background alpha inversion mode.This parameter can be one value of ***DMA2D_LL_EC_ALPHA_INVERSION***. This parameter can be modified afterwards using unitary functions **LL_DMA2D_FGND_SetAlphaInvMode()** for foreground layer,**LL_DMA2D_BGND_SetAlphaInvMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::RBSwapMode***
  Specifies the foreground or background Red Blue swap mode. This parameter can be one value of ***DMA2D_LL_EC_RED_BLUE_SWAP*** .This parameter can be modified afterwards using unitary functions**LL_DMA2D_FGND_SetRBSwapMode()** for foreground layer,**LL_DMA2D_BGND_SetRBSwapMode()** for background layer.

80.1.3 LL_DMA2D_ColorTypeDef

Data Fields

- ***uint32_t ColorMode***
- ***uint32_t OutputBlue***
- ***uint32_t OutputGreen***
- ***uint32_t OutputRed***
- ***uint32_t OutputAlpha***

Field Documentation

- ***uint32_t LL_DMA2D_ColorTypeDef::ColorMode***
  Specifies the color format of the output image.This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_COLOR_MODE***. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColorMode()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputBlue***
  Specifies the Blue value of the output image.This parameter must be a number

between Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.

- ***uint32_t LL_DMA2D_ColorTypeDef::OutputGreen***
  Specifies the Green value of the output image.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x3F if RGB565 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputRed***
  Specifies the Red value of the output image.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputAlpha***
  Specifies the Alpha channel of the output image.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x01 if ARGB1555 color mode is selected.This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.This parameter is not considered if RGB888 or RGB565 color mode is selected. This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.

80.2 DMA2D Firmware driver API description

80.2.1 Detailed description of functions

LL_DMA2D_Start

Function name	**__STATIC_INLINE void LL_DMA2D_Start (DMA2D_TypeDef * DMA2Dx)**
Function description	Start a DMA2D transfer.

Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR START LL_DMA2D_Start

LL_DMA2D_IsTransferOngoing

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsTransferOngoing (DMA2D_TypeDef * DMA2Dx)**
Function description	Indicate if a DMA2D transfer is ongoing.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR START LL_DMA2D_IsTransferOngoing

LL_DMA2D_Suspend

Function name	**__STATIC_INLINE void LL_DMA2D_Suspend (DMA2D_TypeDef * DMA2Dx)**
Function description	Suspend DMA2D transfer.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Notes	• This API can be used to suspend automatic foreground or background CLUT loading.
Reference Manual to LL API cross reference:	• CR SUSP LL_DMA2D_Suspend

LL_DMA2D_Resume

Function name	**__STATIC_INLINE void LL_DMA2D_Resume (DMA2D_TypeDef * DMA2Dx)**
Function description	Resume DMA2D transfer.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Notes	• This API can be used to resume automatic foreground or background CLUT loading.
Reference Manual to LL API cross reference:	• CR SUSP LL_DMA2D_Resume

LL_DMA2D_IsSuspended

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsSuspended (DMA2D_TypeDef * DMA2Dx)**
Function description	Indicate if DMA2D transfer is suspended.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Notes	• This API can be used to indicate whether or not automatic foreground or background CLUT loading is suspended.
Reference Manual to LL API cross reference:	• CR SUSP LL_DMA2D_IsSuspended

LL_DMA2D_Abort

Function name	**__STATIC_INLINE void LL_DMA2D_Abort (DMA2D_TypeDef * DMA2Dx)**
Function description	Abort DMA2D transfer.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Notes	• This API can be used to abort automatic foreground or background CLUT loading.
Reference Manual to LL API cross reference:	• CR ABORT LL_DMA2D_Abort

LL_DMA2D_IsAborted

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsAborted (DMA2D_TypeDef * DMA2Dx)**
Function description	Indicate if DMA2D transfer is aborted.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Notes	• This API can be used to indicate whether or not automatic foreground or background CLUT loading is aborted.
Reference Manual to LL API cross reference:	• CR ABORT LL_DMA2D_IsAborted

LL_DMA2D_SetMode

Function name	**__STATIC_INLINE void LL_DMA2D_SetMode (DMA2D_TypeDef * DMA2Dx, uint32_t Mode)**
Function description	Set DMA2D mode.

Parameters	• **DMA2Dx:** DMA2D Instance • **Mode:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_DMA2D_MODE_M2M – LL_DMA2D_MODE_M2M_PFC – LL_DMA2D_MODE_M2M_BLEND – LL_DMA2D_MODE_R2M – LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG (*) – LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR MODE LL_DMA2D_SetMode

LL_DMA2D_GetMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_DMA2D_MODE_M2M – LL_DMA2D_MODE_M2M_PFC – LL_DMA2D_MODE_M2M_BLEND – LL_DMA2D_MODE_R2M – LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG (*) – LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG (*)
Reference Manual to LL API cross reference:	• CR MODE LL_DMA2D_GetMode

LL_DMA2D_SetOutputColorMode

Function name	**__STATIC_INLINE void LL_DMA2D_SetOutputColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)**
Function description	Set DMA2D output color mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **ColorMode:** This parameter can be one of the following values: – LL_DMA2D_OUTPUT_MODE_ARGB8888 – LL_DMA2D_OUTPUT_MODE_RGB888 – LL_DMA2D_OUTPUT_MODE_RGB565 – LL_DMA2D_OUTPUT_MODE_ARGB1555 – LL_DMA2D_OUTPUT_MODE_ARGB4444
Return values	• **None:**
Reference Manual to	• OPFCCR CM LL_DMA2D_SetOutputColorMode

LL API cross reference:

LL_DMA2D_GetOutputColorMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetOutputColorMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D output color mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_OUTPUT_MODE_ARGB8888 – LL_DMA2D_OUTPUT_MODE_RGB888 – LL_DMA2D_OUTPUT_MODE_RGB565 – LL_DMA2D_OUTPUT_MODE_ARGB1555 – LL_DMA2D_OUTPUT_MODE_ARGB4444
Reference Manual to LL API cross reference:	• OPFCCR CM LL_DMA2D_GetOutputColorMode

LL_DMA2D_SetOutputRBSwapMode

Function name	**__STATIC_INLINE void LL_DMA2D_SetOutputRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)**
Function description	Set DMA2D output Red Blue swap mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **RBSwapMode:** This parameter can be one of the following values: – LL_DMA2D_RB_MODE_REGULAR – LL_DMA2D_RB_MODE_SWAP
Return values	• **None:**
Reference Manual to LL API cross reference:	• OPFCCR RBS LL_DMA2D_SetOutputRBSwapMode

LL_DMA2D_GetOutputRBSwapMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetOutputRBSwapMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D output Red Blue swap mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_RB_MODE_REGULAR – LL_DMA2D_RB_MODE_SWAP
Reference Manual to LL API cross reference:	• OPFCCR RBS LL_DMA2D_GetOutputRBSwapMode

LL_DMA2D_SetOutputAlphaInvMode

Function name	**__STATIC_INLINE void LL_DMA2D_SetOutputAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)**
Function description	Set DMA2D output alpha inversion mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **AlphaInversionMode:** This parameter can be one of the following values: – LL_DMA2D_ALPHA_REGULAR – LL_DMA2D_ALPHA_INVERTED
Return values	• **None:**
Reference Manual to LL API cross reference:	• OPFCCR AI LL_DMA2D_SetOutputAlphaInvMode

LL_DMA2D_GetOutputAlphaInvMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetOutputAlphaInvMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D output alpha inversion mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_ALPHA_REGULAR – LL_DMA2D_ALPHA_INVERTED
Reference Manual to LL API cross reference:	• OPFCCR AI LL_DMA2D_GetOutputAlphaInvMode

LL_DMA2D_SetOutputSwapMode

Function name	**__STATIC_INLINE void LL_DMA2D_SetOutputSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t OutputSwapMode)**
Function description	Set DMA2D output swap mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **OutputSwapMode:** This parameter can be one of the following values: – LL_DMA2D_SWAP_MODE_REGULAR – LL_DMA2D_SWAP_MODE_TWO_BY_TWO
Return values	• **None:**
Reference Manual to LL API cross reference:	• OPFCCR SB LL_DMA2D_SetOutputSwapMode

LL_DMA2D_GetOutputSwapMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetOutputSwapMode (DMA2D_TypeDef * DMA2Dx)**

Function description	Return DMA2D output swap mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_SWAP_MODE_REGULAR – LL_DMA2D_SWAP_MODE_TWO_BY_TWO
Reference Manual to LL API cross reference:	• OPFCCR SB LL_DMA2D_GetOutputSwapMode

LL_DMA2D_SetLineOffsetMode

Function name	**__STATIC_INLINE void LL_DMA2D_SetLineOffsetMode (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffsetMode)**
Function description	Set DMA2D line offset mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **LineOffsetMode:** This parameter can be one of the following values: – LL_DMA2D_LINE_OFFSET_PIXELS – LL_DMA2D_LINE_OFFSET_BYTES
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR LOM LL_DMA2D_SetLineOffsetMode

LL_DMA2D_GetLineOffsetMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetLineOffsetMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D line offset mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_LINE_OFFSET_PIXELS – LL_DMA2D_LINE_OFFSET_BYTES
Reference Manual to LL API cross reference:	• CR LOM LL_DMA2D_GetLineOffsetMode

LL_DMA2D_SetLineOffset

Function name	**__STATIC_INLINE void LL_DMA2D_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)**
Function description	Set DMA2D line offset, expressed on 14 bits ([13:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **LineOffset:** Value between Min_Data=0 and Max_Data=0x3FFF
Return values	• **None:**

Reference Manual to LL API cross reference:	• OOR LO LL_DMA2D_SetLineOffset

LL_DMA2D_GetLineOffset

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetLineOffset (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D line offset, expressed on 14 bits ([13:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Line:** offset value between Min_Data=0 and Max_Data=0x3FFF
Reference Manual to LL API cross reference:	• OOR LO LL_DMA2D_GetLineOffset

LL_DMA2D_SetNbrOfPixelsPerLines

Function name	**__STATIC_INLINE void LL_DMA2D_SetNbrOfPixelsPerLines (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfPixelsPerLines)**
Function description	Set DMA2D number of pixels per lines, expressed on 14 bits ([13:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **NbrOfPixelsPerLines:** Value between Min_Data=0 and Max_Data=0x3FFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• NLR PL LL_DMA2D_SetNbrOfPixelsPerLines

LL_DMA2D_GetNbrOfPixelsPerLines

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetNbrOfPixelsPerLines (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D number of pixels per lines, expressed on 14 bits ([13:0] bits)
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Number:** of pixels per lines value between Min_Data=0 and Max_Data=0x3FFF
Reference Manual to LL API cross reference:	• NLR PL LL_DMA2D_GetNbrOfPixelsPerLines

LL_DMA2D_SetNbrOfLines

Function name	**__STATIC_INLINE void LL_DMA2D_SetNbrOfLines (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfLines)**

Function description	Set DMA2D number of lines, expressed on 16 bits ([15:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **NbrOfLines:** Value between Min_Data=0 and Max_Data=0xFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• NLR NL LL_DMA2D_SetNbrOfLines

LL_DMA2D_GetNbrOfLines

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetNbrOfLines (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D number of lines, expressed on 16 bits ([15:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Number:** of lines value between Min_Data=0 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• NLR NL LL_DMA2D_GetNbrOfLines

LL_DMA2D_SetOutputMemAddr

Function name	**__STATIC_INLINE void LL_DMA2D_SetOutputMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t OutputMemoryAddress)**
Function description	Set DMA2D output memory address, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **OutputMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• OMAR MA LL_DMA2D_SetOutputMemAddr

LL_DMA2D_GetOutputMemAddr

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetOutputMemAddr (DMA2D_TypeDef * DMA2Dx)**
Function description	Get DMA2D output memory address, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Output:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF
Reference Manual to LL API cross	• OMAR MA LL_DMA2D_GetOutputMemAddr

reference:

LL_DMA2D_SetOutputColor

Function name	**__STATIC_INLINE void LL_DMA2D_SetOutputColor (DMA2D_TypeDef * DMA2Dx, uint32_t OutputColor)**
Function description	Set DMA2D output color, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **OutputColor:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF
Return values	• **None:**
Notes	• Output color format depends on output color mode, ARGB8888, RGB888, RGB565, ARGB1555 or ARGB4444. • LL_DMA2D_ConfigOutputColor() API may be used instead if colors values formatting with respect to color mode is not done by the user code.
Reference Manual to LL API cross reference:	• OCOLR BLUE LL_DMA2D_SetOutputColor • OCOLR GREEN LL_DMA2D_SetOutputColor • OCOLR RED LL_DMA2D_SetOutputColor • OCOLR ALPHA LL_DMA2D_SetOutputColor

LL_DMA2D_GetOutputColor

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetOutputColor (DMA2D_TypeDef * DMA2Dx)**
Function description	Get DMA2D output color, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Output:** color value between Min_Data=0 and Max_Data=0xFFFFFFFF
Notes	• Alpha channel and red, green, blue color values must be retrieved from the returned value based on the output color mode (ARGB8888, RGB888, RGB565, ARGB1555 or ARGB4444) as set by LL_DMA2D_SetOutputColorMode.
Reference Manual to LL API cross reference:	• OCOLR BLUE LL_DMA2D_GetOutputColor • OCOLR GREEN LL_DMA2D_GetOutputColor • OCOLR RED LL_DMA2D_GetOutputColor • OCOLR ALPHA LL_DMA2D_GetOutputColor

LL_DMA2D_SetLineWatermark

Function name	**__STATIC_INLINE void LL_DMA2D_SetLineWatermark (DMA2D_TypeDef * DMA2Dx, uint32_t LineWatermark)**
Function description	Set DMA2D line watermark, expressed on 16 bits ([15:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **LineWatermark:** Value between Min_Data=0 and Max_Data=0xFFFF

Return values	• **None:**
Reference Manual to LL API cross reference:	• LWR LW LL_DMA2D_SetLineWatermark

LL_DMA2D_GetLineWatermark

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetLineWatermark (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D line watermark, expressed on 16 bits ([15:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Line:** watermark value between Min_Data=0 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• LWR LW LL_DMA2D_GetLineWatermark

LL_DMA2D_SetDeadTime

Function name	**__STATIC_INLINE void LL_DMA2D_SetDeadTime (DMA2D_TypeDef * DMA2Dx, uint32_t DeadTime)**
Function description	Set DMA2D dead time, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **DeadTime:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• AMTCR DT LL_DMA2D_SetDeadTime

LL_DMA2D_GetDeadTime

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_GetDeadTime (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D dead time, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Dead:** time value between Min_Data=0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• AMTCR DT LL_DMA2D_GetDeadTime

LL_DMA2D_EnableDeadTime

Function name	**__STATIC_INLINE void LL_DMA2D_EnableDeadTime (DMA2D_TypeDef * DMA2Dx)**
Function description	Enable DMA2D dead time functionality.

Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• AMTCR EN LL_DMA2D_EnableDeadTime

LL_DMA2D_DisableDeadTime

Function name	**__STATIC_INLINE void LL_DMA2D_DisableDeadTime (DMA2D_TypeDef * DMA2Dx)**
Function description	Disable DMA2D dead time functionality.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• AMTCR EN LL_DMA2D_DisableDeadTime

LL_DMA2D_IsEnabledDeadTime

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledDeadTime (DMA2D_TypeDef * DMA2Dx)**
Function description	Indicate if DMA2D dead time functionality is enabled.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• AMTCR EN LL_DMA2D_IsEnabledDeadTime

LL_DMA2D_FGND_SetMemAddr

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t MemoryAddress)**
Function description	Set DMA2D foreground memory address, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **MemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGMAR MA LL_DMA2D_FGND_SetMemAddr

LL_DMA2D_FGND_GetMemAddr

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetMemAddr (DMA2D_TypeDef * DMA2Dx)**

Function description	Get DMA2D foreground memory address, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Foreground:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF
Reference Manual to LL API cross reference:	• FGMAR MA LL_DMA2D_FGND_GetMemAddr

LL_DMA2D_FGND_EnableCLUTLoad

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_EnableCLUTLoad (DMA2D_TypeDef * DMA2Dx)**
Function description	Enable DMA2D foreground CLUT loading.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGPFCCR START LL_DMA2D_FGND_EnableCLUTLoad

LL_DMA2D_FGND_IsEnabledCLUTLoad

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_IsEnabledCLUTLoad (DMA2D_TypeDef * DMA2Dx)**
Function description	Indicate if DMA2D foreground CLUT loading is enabled.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• FGPFCCR START LL_DMA2D_FGND_IsEnabledCLUTLoad

LL_DMA2D_FGND_SetColorMode

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)**
Function description	Set DMA2D foreground color mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **ColorMode:** This parameter can be one of the following values: – LL_DMA2D_INPUT_MODE_ARGB8888 – LL_DMA2D_INPUT_MODE_RGB888 – LL_DMA2D_INPUT_MODE_RGB565 – LL_DMA2D_INPUT_MODE_ARGB1555 – LL_DMA2D_INPUT_MODE_ARGB4444 – LL_DMA2D_INPUT_MODE_L8 – LL_DMA2D_INPUT_MODE_AL44

- – LL_DMA2D_INPUT_MODE_AL88
- – LL_DMA2D_INPUT_MODE_L4
- – LL_DMA2D_INPUT_MODE_A8
- – LL_DMA2D_INPUT_MODE_A4

Return values	• **None:**
Reference Manual to LL API cross reference:	• FGPFCCR CM LL_DMA2D_FGND_SetColorMode

LL_DMA2D_FGND_GetColorMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetColorMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D foreground color mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_INPUT_MODE_ARGB8888 – LL_DMA2D_INPUT_MODE_RGB888 – LL_DMA2D_INPUT_MODE_RGB565 – LL_DMA2D_INPUT_MODE_ARGB1555 – LL_DMA2D_INPUT_MODE_ARGB4444 – LL_DMA2D_INPUT_MODE_L8 – LL_DMA2D_INPUT_MODE_AL44 – LL_DMA2D_INPUT_MODE_AL88 – LL_DMA2D_INPUT_MODE_L4 – LL_DMA2D_INPUT_MODE_A8 – LL_DMA2D_INPUT_MODE_A4
Reference Manual to LL API cross reference:	• FGPFCCR CM LL_DMA2D_FGND_GetColorMode

LL_DMA2D_FGND_SetAlphaMode

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetAlphaMode (DMA2D_TypeDef * DMA2Dx, uint32_t AphaMode)**
Function description	Set DMA2D foreground alpha mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **AphaMode:** This parameter can be one of the following values: – LL_DMA2D_ALPHA_MODE_NO_MODIF – LL_DMA2D_ALPHA_MODE_REPLACE – LL_DMA2D_ALPHA_MODE_COMBINE
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGPFCCR AM LL_DMA2D_FGND_SetAlphaMode

LL_DMA2D_FGND_GetAlphaMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlphaMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D foreground alpha mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_ALPHA_MODE_NO_MODIF – LL_DMA2D_ALPHA_MODE_REPLACE – LL_DMA2D_ALPHA_MODE_COMBINE
Reference Manual to LL API cross reference:	• FGPFCCR AM LL_DMA2D_FGND_GetAlphaMode

LL_DMA2D_FGND_SetAlpha

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetAlpha (DMA2D_TypeDef * DMA2Dx, uint32_t Alpha)**
Function description	Set DMA2D foreground alpha value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **Alpha:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGPFCCR ALPHA LL_DMA2D_FGND_SetAlpha

LL_DMA2D_FGND_GetAlpha

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlpha (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D foreground alpha value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Alpha:** value between Min_Data=0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• FGPFCCR ALPHA LL_DMA2D_FGND_GetAlpha

LL_DMA2D_FGND_SetRBSwapMode

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)**
Function description	Set DMA2D foreground Red Blue swap mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **RBSwapMode:** This parameter can be one of the following values:

	– LL_DMA2D_RB_MODE_REGULAR – LL_DMA2D_RB_MODE_SWAP
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGPFCCR RBS LL_DMA2D_FGND_SetRBSwapMode

LL_DMA2D_FGND_GetRBSwapMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetRBSwapMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D foreground Red Blue swap mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_RB_MODE_REGULAR – LL_DMA2D_RB_MODE_SWAP
Reference Manual to LL API cross reference:	• FGPFCCR RBS LL_DMA2D_FGND_GetRBSwapMode

LL_DMA2D_FGND_SetAlphaInvMode

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)**
Function description	Set DMA2D foreground alpha inversion mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **AlphaInversionMode:** This parameter can be one of the following values: – LL_DMA2D_ALPHA_REGULAR – LL_DMA2D_ALPHA_INVERTED
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGPFCCR AI LL_DMA2D_FGND_SetAlphaInvMode

LL_DMA2D_FGND_GetAlphaInvMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlphaInvMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D foreground alpha inversion mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_ALPHA_REGULAR – LL_DMA2D_ALPHA_INVERTED

Reference Manual to LL API cross reference:	• FGPFCCR AI LL_DMA2D_FGND_GetAlphaInvMode

LL_DMA2D_FGND_SetLineOffset

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)**
Function description	Set DMA2D foreground line offset, expressed on 14 bits ([13:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **LineOffset:** Value between Min_Data=0 and Max_Data=0x3FF
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGOR LO LL_DMA2D_FGND_SetLineOffset

LL_DMA2D_FGND_GetLineOffset

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetLineOffset (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D foreground line offset, expressed on 14 bits ([13:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Foreground:** line offset value between Min_Data=0 and Max_Data=0x3FF
Reference Manual to LL API cross reference:	• FGOR LO LL_DMA2D_FGND_GetLineOffset

LL_DMA2D_FGND_SetColor

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red, uint32_t Green, uint32_t Blue)**
Function description	Set DMA2D foreground color values, expressed on 24 bits ([23:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **Red:** Value between Min_Data=0 and Max_Data=0xFF • **Green:** Value between Min_Data=0 and Max_Data=0xFF • **Blue:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGCOLR RED LL_DMA2D_FGND_SetColor • FGCOLR GREEN LL_DMA2D_FGND_SetColor • FGCOLR BLUE LL_DMA2D_FGND_SetColor

LL_DMA2D_FGND_SetRedColor

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red)**
Function description	Set DMA2D foreground red color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **Red:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGCOLR RED LL_DMA2D_FGND_SetRedColor

LL_DMA2D_FGND_GetRedColor

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetRedColor (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D foreground red color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Red:** color value between Min_Data=0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• FGCOLR RED LL_DMA2D_FGND_GetRedColor

LL_DMA2D_FGND_SetGreenColor

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t Green)**
Function description	Set DMA2D foreground green color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **Green:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGCOLR GREEN LL_DMA2D_FGND_SetGreenColor

LL_DMA2D_FGND_GetGreenColor

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetGreenColor (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D foreground green color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Green:** color value between Min_Data=0 and

		Max_Data=0xFF
Reference Manual to LL API cross reference:	•	FGCOLR GREEN LL_DMA2D_FGND_GetGreenColor

LL_DMA2D_FGND_SetBlueColor

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t Blue)**
Function description	Set DMA2D foreground blue color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **Blue:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGCOLR BLUE LL_DMA2D_FGND_SetBlueColor

LL_DMA2D_FGND_GetBlueColor

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetBlueColor (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D foreground blue color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Blue:** color value between Min_Data=0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• FGCOLR BLUE LL_DMA2D_FGND_GetBlueColor

LL_DMA2D_FGND_SetCLUTMemAddr

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTMemoryAddress)**
Function description	Set DMA2D foreground CLUT memory address, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **CLUTMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGCMAR MA LL_DMA2D_FGND_SetCLUTMemAddr

LL_DMA2D_FGND_GetCLUTMemAddr

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTMemAddr (DMA2D_TypeDef ***

DMA2Dx)

Function description	Get DMA2D foreground CLUT memory address, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Foreground:** CLUT memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF
Reference Manual to LL API cross reference:	• FGCMAR MA LL_DMA2D_FGND_GetCLUTMemAddr

LL_DMA2D_FGND_SetCLUTSize

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTSize (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTSize)**
Function description	Set DMA2D foreground CLUT size, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **CLUTSize:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGPFCCR CS LL_DMA2D_FGND_SetCLUTSize

LL_DMA2D_FGND_GetCLUTSize

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTSize (DMA2D_TypeDef * DMA2Dx)**
Function description	Get DMA2D foreground CLUT size, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Foreground:** CLUT size value between Min_Data=0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• FGPFCCR CS LL_DMA2D_FGND_GetCLUTSize

LL_DMA2D_FGND_SetCLUTColorMode

Function name	**__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTColorMode)**
Function description	Set DMA2D foreground CLUT color mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **CLUTColorMode:** This parameter can be one of the following values: – LL_DMA2D_CLUT_COLOR_MODE_ARGB8888

	– LL_DMA2D_CLUT_COLOR_MODE_RGB888
Return values	• **None:**
Reference Manual to LL API cross reference:	• FGPFCCR CCM LL_DMA2D_FGND_SetCLUTColorMode

LL_DMA2D_FGND_GetCLUTColorMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTColorMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D foreground CLUT color mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_CLUT_COLOR_MODE_ARGB8888 – LL_DMA2D_CLUT_COLOR_MODE_RGB888
Reference Manual to LL API cross reference:	• FGPFCCR CCM LL_DMA2D_FGND_GetCLUTColorMode

LL_DMA2D_BGND_SetMemAddr

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t MemoryAddress)**
Function description	Set DMA2D background memory address, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **MemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGMAR MA LL_DMA2D_BGND_SetMemAddr

LL_DMA2D_BGND_GetMemAddr

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetMemAddr (DMA2D_TypeDef * DMA2Dx)**
Function description	Get DMA2D background memory address, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Background:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF
Reference Manual to LL API cross reference:	• BGMAR MA LL_DMA2D_BGND_GetMemAddr

LL_DMA2D_BGND_EnableCLUTLoad

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_EnableCLUTLoad (DMA2D_TypeDef * DMA2Dx)**
Function description	Enable DMA2D background CLUT loading.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGPFCCR START LL_DMA2D_BGND_EnableCLUTLoad

LL_DMA2D_BGND_IsEnabledCLUTLoad

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_IsEnabledCLUTLoad (DMA2D_TypeDef * DMA2Dx)**
Function description	Indicate if DMA2D background CLUT loading is enabled.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• BGPFCCR START LL_DMA2D_BGND_IsEnabledCLUTLoad

LL_DMA2D_BGND_SetColorMode

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)**
Function description	Set DMA2D background color mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **ColorMode:** This parameter can be one of the following values: – LL_DMA2D_INPUT_MODE_ARGB8888 – LL_DMA2D_INPUT_MODE_RGB888 – LL_DMA2D_INPUT_MODE_RGB565 – LL_DMA2D_INPUT_MODE_ARGB1555 – LL_DMA2D_INPUT_MODE_ARGB4444 – LL_DMA2D_INPUT_MODE_L8 – LL_DMA2D_INPUT_MODE_AL44 – LL_DMA2D_INPUT_MODE_AL88 – LL_DMA2D_INPUT_MODE_L4 – LL_DMA2D_INPUT_MODE_A8 – LL_DMA2D_INPUT_MODE_A4
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGPFCCR CM LL_DMA2D_BGND_SetColorMode

LL_DMA2D_BGND_GetColorMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetColorMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D background color mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_INPUT_MODE_ARGB8888 – LL_DMA2D_INPUT_MODE_RGB888 – LL_DMA2D_INPUT_MODE_RGB565 – LL_DMA2D_INPUT_MODE_ARGB1555 – LL_DMA2D_INPUT_MODE_ARGB4444 – LL_DMA2D_INPUT_MODE_L8 – LL_DMA2D_INPUT_MODE_AL44 – LL_DMA2D_INPUT_MODE_AL88 – LL_DMA2D_INPUT_MODE_L4 – LL_DMA2D_INPUT_MODE_A8 – LL_DMA2D_INPUT_MODE_A4
Reference Manual to LL API cross reference:	• BGPFCCR CM LL_DMA2D_BGND_GetColorMode

LL_DMA2D_BGND_SetAlphaMode

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetAlphaMode (DMA2D_TypeDef * DMA2Dx, uint32_t AphaMode)**
Function description	Set DMA2D background alpha mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **AphaMode:** This parameter can be one of the following values: – LL_DMA2D_ALPHA_MODE_NO_MODIF – LL_DMA2D_ALPHA_MODE_REPLACE – LL_DMA2D_ALPHA_MODE_COMBINE
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGPFCCR AM LL_DMA2D_BGND_SetAlphaMode

LL_DMA2D_BGND_GetAlphaMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlphaMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D background alpha mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_ALPHA_MODE_NO_MODIF – LL_DMA2D_ALPHA_MODE_REPLACE – LL_DMA2D_ALPHA_MODE_COMBINE

Reference Manual to LL API cross reference:	• BGPFCCR AM LL_DMA2D_BGND_GetAlphaMode

LL_DMA2D_BGND_SetAlpha

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetAlpha (DMA2D_TypeDef * DMA2Dx, uint32_t Alpha)**
Function description	Set DMA2D background alpha value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **Alpha:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGPFCCR ALPHA LL_DMA2D_BGND_SetAlpha

LL_DMA2D_BGND_GetAlpha

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlpha (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D background alpha value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Alpha:** value between Min_Data=0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• BGPFCCR ALPHA LL_DMA2D_BGND_GetAlpha

LL_DMA2D_BGND_SetRBSwapMode

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)**
Function description	Set DMA2D background Red Blue swap mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **RBSwapMode:** This parameter can be one of the following values: – LL_DMA2D_RB_MODE_REGULAR – LL_DMA2D_RB_MODE_SWAP
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGPFCCR RBS LL_DMA2D_BGND_SetRBSwapMode

LL_DMA2D_BGND_GetRBSwapMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetRBSwapMode (DMA2D_TypeDef ***

	DMA2Dx)
Function description	Return DMA2D background Red Blue swap mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_RB_MODE_REGULAR – LL_DMA2D_RB_MODE_SWAP
Reference Manual to LL API cross reference:	• BGPFCCR RBS LL_DMA2D_BGND_GetRBSwapMode

LL_DMA2D_BGND_SetAlphaInvMode

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)**
Function description	Set DMA2D background alpha inversion mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **AlphaInversionMode:** This parameter can be one of the following values: – LL_DMA2D_ALPHA_REGULAR – LL_DMA2D_ALPHA_INVERTED
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGPFCCR AI LL_DMA2D_BGND_SetAlphaInvMode

LL_DMA2D_BGND_GetAlphaInvMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlphaInvMode (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D background alpha inversion mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_ALPHA_REGULAR – LL_DMA2D_ALPHA_INVERTED
Reference Manual to LL API cross reference:	• BGPFCCR AI LL_DMA2D_BGND_GetAlphaInvMode

LL_DMA2D_BGND_SetLineOffset

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)**
Function description	Set DMA2D background line offset, expressed on 14 bits ([13:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **LineOffset:** Value between Min_Data=0 and

	Max_Data=0x3FF
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGOR LO LL_DMA2D_BGND_SetLineOffset

LL_DMA2D_BGND_GetLineOffset

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetLineOffset (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D background line offset, expressed on 14 bits ([13:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Background:** line offset value between Min_Data=0 and Max_Data=0x3FF
Reference Manual to LL API cross reference:	• BGOR LO LL_DMA2D_BGND_GetLineOffset

LL_DMA2D_BGND_SetColor

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red, uint32_t Green, uint32_t Blue)**
Function description	Set DMA2D background color values, expressed on 24 bits ([23:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **Red:** Value between Min_Data=0 and Max_Data=0xFF • **Green:** Value between Min_Data=0 and Max_Data=0xFF • **Blue:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGCOLR RED LL_DMA2D_BGND_SetColor • BGCOLR GREEN LL_DMA2D_BGND_SetColor • BGCOLR BLUE LL_DMA2D_BGND_SetColor

LL_DMA2D_BGND_SetRedColor

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red)**
Function description	Set DMA2D background red color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **Red:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross	• BGCOLR RED LL_DMA2D_BGND_SetRedColor

reference:

LL_DMA2D_BGND_GetRedColor

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetRedColor (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D background red color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Red:** color value between Min_Data=0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• BGCOLR RED LL_DMA2D_BGND_GetRedColor

LL_DMA2D_BGND_SetGreenColor

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t Green)**
Function description	Set DMA2D background green color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **Green:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGCOLR GREEN LL_DMA2D_BGND_SetGreenColor

LL_DMA2D_BGND_GetGreenColor

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetGreenColor (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D background green color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Green:** color value between Min_Data=0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• BGCOLR GREEN LL_DMA2D_BGND_GetGreenColor

LL_DMA2D_BGND_SetBlueColor

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t Blue)**
Function description	Set DMA2D background blue color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance

	• **Blue:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGCOLR BLUE LL_DMA2D_BGND_SetBlueColor

LL_DMA2D_BGND_GetBlueColor

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetBlueColor (DMA2D_TypeDef * DMA2Dx)**
Function description	Return DMA2D background blue color value, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Blue:** color value between Min_Data=0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• BGCOLR BLUE LL_DMA2D_BGND_GetBlueColor

LL_DMA2D_BGND_SetCLUTMemAddr

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTMemoryAddress)**
Function description	Set DMA2D background CLUT memory address, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **CLUTMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGCMAR MA LL_DMA2D_BGND_SetCLUTMemAddr

LL_DMA2D_BGND_GetCLUTMemAddr

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx)**
Function description	Get DMA2D background CLUT memory address, expressed on 32 bits ([31:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Background:** CLUT memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF
Reference Manual to LL API cross reference:	• BGCMAR MA LL_DMA2D_BGND_GetCLUTMemAddr

LL_DMA2D_BGND_SetCLUTSize

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTSize (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTSize)**
Function description	Set DMA2D background CLUT size, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance • **CLUTSize:** Value between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGPFCCR CS LL_DMA2D_BGND_SetCLUTSize

LL_DMA2D_BGND_GetCLUTSize

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTSize (DMA2D_TypeDef * DMA2Dx)**
Function description	Get DMA2D background CLUT size, expressed on 8 bits ([7:0] bits).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Background:** CLUT size value between Min_Data=0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• BGPFCCR CS LL_DMA2D_BGND_GetCLUTSize

LL_DMA2D_BGND_SetCLUTColorMode

Function name	**__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTColorMode)**
Function description	Set DMA2D background CLUT color mode.
Parameters	• **DMA2Dx:** DMA2D Instance • **CLUTColorMode:** This parameter can be one of the following values: – LL_DMA2D_CLUT_COLOR_MODE_ARGB8888 – LL_DMA2D_CLUT_COLOR_MODE_RGB888
Return values	• **None:**
Reference Manual to LL API cross reference:	• BGPFCCR CCM LL_DMA2D_BGND_SetCLUTColorMode

LL_DMA2D_BGND_GetCLUTColorMode

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTColorMode (DMA2D_TypeDef * DMA2Dx)**

Function description	Return DMA2D background CLUT color mode.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **Returned:** value can be one of the following values: – LL_DMA2D_CLUT_COLOR_MODE_ARGB8888 – LL_DMA2D_CLUT_COLOR_MODE_RGB888
Reference Manual to LL API cross reference:	• BGPFCCR CCM LL_DMA2D_BGND_GetCLUTColorMode

LL_DMA2D_IsActiveFlag_CE

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CE (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D Configuration Error Interrupt Flag is set or not.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CEIF LL_DMA2D_IsActiveFlag_CE

LL_DMA2D_IsActiveFlag_CTC

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CTC (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D CLUT Transfer Complete Interrupt Flag is set or not.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CTCIF LL_DMA2D_IsActiveFlag_CTC

LL_DMA2D_IsActiveFlag_CAE

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CAE (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D CLUT Access Error Interrupt Flag is set or not.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CAEIF LL_DMA2D_IsActiveFlag_CAE

LL_DMA2D_IsActiveFlag_TW

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TW (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D Transfer Watermark Interrupt Flag is set or not.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TWIF LL_DMA2D_IsActiveFlag_TW

LL_DMA2D_IsActiveFlag_TC

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TC (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D Transfer Complete Interrupt Flag is set or not.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF LL_DMA2D_IsActiveFlag_TC

LL_DMA2D_IsActiveFlag_TE

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TE (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D Transfer Error Interrupt Flag is set or not.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF LL_DMA2D_IsActiveFlag_TE

LL_DMA2D_ClearFlag_CE

Function name	**__STATIC_INLINE void LL_DMA2D_ClearFlag_CE (DMA2D_TypeDef * DMA2Dx)**
Function description	Clear DMA2D Configuration Error Interrupt Flag.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CCEIF LL_DMA2D_ClearFlag_CE

LL_DMA2D_ClearFlag_CTC

Function name	**__STATIC_INLINE void LL_DMA2D_ClearFlag_CTC (DMA2D_TypeDef * DMA2Dx)**
Function description	Clear DMA2D CLUT Transfer Complete Interrupt Flag.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CCTCIF LL_DMA2D_ClearFlag_CTC

LL_DMA2D_ClearFlag_CAE

Function name	**__STATIC_INLINE void LL_DMA2D_ClearFlag_CAE (DMA2D_TypeDef * DMA2Dx)**
Function description	Clear DMA2D CLUT Access Error Interrupt Flag.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CAECIF LL_DMA2D_ClearFlag_CAE

LL_DMA2D_ClearFlag_TW

Function name	**__STATIC_INLINE void LL_DMA2D_ClearFlag_TW (DMA2D_TypeDef * DMA2Dx)**
Function description	Clear DMA2D Transfer Watermark Interrupt Flag.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTWIF LL_DMA2D_ClearFlag_TW

LL_DMA2D_ClearFlag_TC

Function name	**__STATIC_INLINE void LL_DMA2D_ClearFlag_TC (DMA2D_TypeDef * DMA2Dx)**
Function description	Clear DMA2D Transfer Complete Interrupt Flag.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF LL_DMA2D_ClearFlag_TC

LL_DMA2D_ClearFlag_TE

Function name	**__STATIC_INLINE void LL_DMA2D_ClearFlag_TE (DMA2D_TypeDef * DMA2Dx)**
Function description	Clear DMA2D Transfer Error Interrupt Flag.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF LL_DMA2D_ClearFlag_TE

LL_DMA2D_EnableIT_CE

Function name	**__STATIC_INLINE void LL_DMA2D_EnableIT_CE (DMA2D_TypeDef * DMA2Dx)**
Function description	Enable Configuration Error Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CEIE LL_DMA2D_EnableIT_CE

LL_DMA2D_EnableIT_CTC

Function name	**__STATIC_INLINE void LL_DMA2D_EnableIT_CTC (DMA2D_TypeDef * DMA2Dx)**
Function description	Enable CLUT Transfer Complete Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CTCIE LL_DMA2D_EnableIT_CTC

LL_DMA2D_EnableIT_CAE

Function name	**__STATIC_INLINE void LL_DMA2D_EnableIT_CAE (DMA2D_TypeDef * DMA2Dx)**
Function description	Enable CLUT Access Error Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CAEIE LL_DMA2D_EnableIT_CAE

LL_DMA2D_EnableIT_TW

Function name	**__STATIC_INLINE void LL_DMA2D_EnableIT_TW (DMA2D_TypeDef * DMA2Dx)**
Function description	Enable Transfer Watermark Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR TWIE LL_DMA2D_EnableIT_TW

LL_DMA2D_EnableIT_TC

Function name	**__STATIC_INLINE void LL_DMA2D_EnableIT_TC (DMA2D_TypeDef * DMA2Dx)**
Function description	Enable Transfer Complete Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR TCIE LL_DMA2D_EnableIT_TC

LL_DMA2D_EnableIT_TE

Function name	**__STATIC_INLINE void LL_DMA2D_EnableIT_TE (DMA2D_TypeDef * DMA2Dx)**
Function description	Enable Transfer Error Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR TEIE LL_DMA2D_EnableIT_TE

LL_DMA2D_DisableIT_CE

Function name	**__STATIC_INLINE void LL_DMA2D_DisableIT_CE (DMA2D_TypeDef * DMA2Dx)**
Function description	Disable Configuration Error Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CEIE LL_DMA2D_DisableIT_CE

LL_DMA2D_DisableIT_CTC

Function name	**__STATIC_INLINE void LL_DMA2D_DisableIT_CTC (DMA2D_TypeDef * DMA2Dx)**
Function description	Disable CLUT Transfer Complete Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CTCIE LL_DMA2D_DisableIT_CTC

LL_DMA2D_DisableIT_CAE

Function name	**__STATIC_INLINE void LL_DMA2D_DisableIT_CAE (DMA2D_TypeDef * DMA2Dx)**
Function description	Disable CLUT Access Error Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CAEIE LL_DMA2D_DisableIT_CAE

LL_DMA2D_DisableIT_TW

Function name	**__STATIC_INLINE void LL_DMA2D_DisableIT_TW (DMA2D_TypeDef * DMA2Dx)**
Function description	Disable Transfer Watermark Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR TWIE LL_DMA2D_DisableIT_TW

LL_DMA2D_DisableIT_TC

Function name	**__STATIC_INLINE void LL_DMA2D_DisableIT_TC (DMA2D_TypeDef * DMA2Dx)**
Function description	Disable Transfer Complete Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR TCIE LL_DMA2D_DisableIT_TC

LL_DMA2D_DisableIT_TE

Function name	**__STATIC_INLINE void LL_DMA2D_DisableIT_TE (DMA2D_TypeDef * DMA2Dx)**
Function description	Disable Transfer Error Interrupt.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR TEIE LL_DMA2D_DisableIT_TE

LL_DMA2D_IsEnabledIT_CE

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CE (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D Configuration Error interrupt source is enabled or disabled.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR CEIE LL_DMA2D_IsEnabledIT_CE

LL_DMA2D_IsEnabledIT_CTC

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CTC (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D CLUT Transfer Complete interrupt source is enabled or disabled.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR CTCIE LL_DMA2D_IsEnabledIT_CTC

LL_DMA2D_IsEnabledIT_CAE

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CAE (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D CLUT Access Error interrupt source is enabled or disabled.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR CAEIE LL_DMA2D_IsEnabledIT_CAE

LL_DMA2D_IsEnabledIT_TW

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TW (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D Transfer Watermark interrupt source is enabled or disabled.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR TWIE LL_DMA2D_IsEnabledIT_TW

LL_DMA2D_IsEnabledIT_TC

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TC (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D Transfer Complete interrupt source is enabled or disabled.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR TCIE LL_DMA2D_IsEnabledIT_TC

LL_DMA2D_IsEnabledIT_TE

Function name	**__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TE (DMA2D_TypeDef * DMA2Dx)**
Function description	Check if the DMA2D Transfer Error interrupt source is enabled or disabled.
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR TEIE LL_DMA2D_IsEnabledIT_TE

LL_DMA2D_DeInit

Function name	**ErrorStatus LL_DMA2D_DeInit (DMA2D_TypeDef * DMA2Dx)**
Function description	De-initialize DMA2D registers (registers restored to their default values).
Parameters	• **DMA2Dx:** DMA2D Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: DMA2D registers are de-initialized – ERROR: DMA2D registers are not de-initialized

LL_DMA2D_Init

Function name	**ErrorStatus LL_DMA2D_Init (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_InitTypeDef * DMA2D_InitStruct)**
Function description	Initialize DMA2D registers according to the specified parameters in DMA2D_InitStruct.
Parameters	• **DMA2Dx:** DMA2D Instance • **DMA2D_InitStruct:** pointer to a LL_DMA2D_InitTypeDef structure that contains the configuration information for the specified DMA2D peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: DMA2D registers are initialized according to DMA2D_InitStruct content – ERROR: Issue occurred during DMA2D registers initialization
Notes	• DMA2D transfers must be disabled to set initialization bits in configuration registers, otherwise ERROR result is returned.

LL_DMA2D_StructInit

Function name	**void LL_DMA2D_StructInit (LL_DMA2D_InitTypeDef * DMA2D_InitStruct)**
Function description	Set each LL_DMA2D_InitTypeDef field to default value.
Parameters	• **DMA2D_InitStruct:** pointer to a LL_DMA2D_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

LL_DMA2D_ConfigLayer

Function name	**void LL_DMA2D_ConfigLayer (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_LayerCfgTypeDef * DMA2D_LayerCfg, uint32_t LayerIdx)**
Function description	Configure the foreground or background according to the specified parameters in the LL_DMA2D_LayerCfgTypeDef structure.
Parameters	• **DMA2Dx:** DMA2D Instance • **DMA2D_LayerCfg:** pointer to a LL_DMA2D_LayerCfgTypeDef structure that contains the configuration information for the specified layer. • **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)
Return values	• **None:**

LL_DMA2D_LayerCfgStructInit

Function name	**void LL_DMA2D_LayerCfgStructInit (LL_DMA2D_LayerCfgTypeDef * DMA2D_LayerCfg)**
Function description	Set each LL_DMA2D_LayerCfgTypeDef field to default value.
Parameters	• **DMA2D_LayerCfg:** pointer to a

	LL_DMA2D_LayerCfgTypeDef structure whose fields will be set to default values.
Return values	• **None:**

LL_DMA2D_ConfigOutputColor

Function name	**void LL_DMA2D_ConfigOutputColor (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_ColorTypeDef * DMA2D_ColorStruct)**
Function description	Initialize DMA2D output color register according to the specified parameters in DMA2D_ColorStruct.
Parameters	• **DMA2Dx:** DMA2D Instance • **DMA2D_ColorStruct:** pointer to a LL_DMA2D_ColorTypeDef structure that contains the color configuration information for the specified DMA2D peripheral.
Return values	• **None:**

LL_DMA2D_GetOutputBlueColor

Function name	**uint32_t LL_DMA2D_GetOutputBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)**
Function description	Return DMA2D output Blue color.
Parameters	• **DMA2Dx:** DMA2D Instance. • **ColorMode:** This parameter can be one of the following values: – LL_DMA2D_OUTPUT_MODE_ARGB8888 – LL_DMA2D_OUTPUT_MODE_RGB888 – LL_DMA2D_OUTPUT_MODE_RGB565 – LL_DMA2D_OUTPUT_MODE_ARGB1555 – LL_DMA2D_OUTPUT_MODE_ARGB4444
Return values	• **Output:** Blue color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputGreenColor

Function name	**uint32_t LL_DMA2D_GetOutputGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)**
Function description	Return DMA2D output Green color.
Parameters	• **DMA2Dx:** DMA2D Instance. • **ColorMode:** This parameter can be one of the following values: – LL_DMA2D_OUTPUT_MODE_ARGB8888 – LL_DMA2D_OUTPUT_MODE_RGB888 – LL_DMA2D_OUTPUT_MODE_RGB565 – LL_DMA2D_OUTPUT_MODE_ARGB1555 – LL_DMA2D_OUTPUT_MODE_ARGB4444
Return values	• **Output:** Green color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputRedColor

Function name	**uint32_t LL_DMA2D_GetOutputRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)**
Function description	Return DMA2D output Red color.
Parameters	• **DMA2Dx:** DMA2D Instance. • **ColorMode:** This parameter can be one of the following values: – LL_DMA2D_OUTPUT_MODE_ARGB8888 – LL_DMA2D_OUTPUT_MODE_RGB888 – LL_DMA2D_OUTPUT_MODE_RGB565 – LL_DMA2D_OUTPUT_MODE_ARGB1555 – LL_DMA2D_OUTPUT_MODE_ARGB4444
Return values	• **Output:** Red color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputAlphaColor

Function name	**uint32_t LL_DMA2D_GetOutputAlphaColor (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)**
Function description	Return DMA2D output Alpha color.
Parameters	• **DMA2Dx:** DMA2D Instance. • **ColorMode:** This parameter can be one of the following values: – LL_DMA2D_OUTPUT_MODE_ARGB8888 – LL_DMA2D_OUTPUT_MODE_RGB888 – LL_DMA2D_OUTPUT_MODE_RGB565 – LL_DMA2D_OUTPUT_MODE_ARGB1555 – LL_DMA2D_OUTPUT_MODE_ARGB4444
Return values	• **Output:** Alpha color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_ConfigSize

Function name	**void LL_DMA2D_ConfigSize (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfLines, uint32_t NbrOfPixelsPerLines)**
Function description	Configure DMA2D transfer size.
Parameters	• **DMA2Dx:** DMA2D Instance • **NbrOfLines:** Value between Min_Data=0 and Max_Data=0xFFFF • **NbrOfPixelsPerLines:** Value between Min_Data=0 and Max_Data=0x3FFF
Return values	• **None:**

80.3 DMA2D Firmware driver defines

80.3.1 DMA2D

Alpha Inversion

LL_DMA2D_ALPHA_REGULAR	Regular alpha
LL_DMA2D_ALPHA_INVERTED	Inverted alpha

Alpha Mode

LL_DMA2D_ALPHA_MODE_NO_MODIF	No modification of the alpha channel value
LL_DMA2D_ALPHA_MODE_REPLACE	Replace original alpha channel value by programmed alpha value
LL_DMA2D_ALPHA_MODE_COMBINE	Replace original alpha channel value by programmed alpha value with original alpha channel value

CLUT Color Mode

LL_DMA2D_CLUT_COLOR_MODE_ARGB8888	ARGB8888
LL_DMA2D_CLUT_COLOR_MODE_RGB888	RGB888

Get Flags Defines

LL_DMA2D_FLAG_CEIF	Configuration Error Interrupt Flag
LL_DMA2D_FLAG_CTCIF	CLUT Transfer Complete Interrupt Flag
LL_DMA2D_FLAG_CAEIF	CLUT Access Error Interrupt Flag
LL_DMA2D_FLAG_TWIF	Transfer Watermark Interrupt Flag
LL_DMA2D_FLAG_TCIF	Transfer Complete Interrupt Flag
LL_DMA2D_FLAG_TEIF	Transfer Error Interrupt Flag

Input Color Mode

LL_DMA2D_INPUT_MODE_ARGB8888	ARGB8888
LL_DMA2D_INPUT_MODE_RGB888	RGB888
LL_DMA2D_INPUT_MODE_RGB565	RGB565
LL_DMA2D_INPUT_MODE_ARGB1555	ARGB1555
LL_DMA2D_INPUT_MODE_ARGB4444	ARGB4444
LL_DMA2D_INPUT_MODE_L8	L8
LL_DMA2D_INPUT_MODE_AL44	AL44
LL_DMA2D_INPUT_MODE_AL88	AL88
LL_DMA2D_INPUT_MODE_L4	L4
LL_DMA2D_INPUT_MODE_A8	A8
LL_DMA2D_INPUT_MODE_A4	A4

IT Defines

LL_DMA2D_IT_CEIE	Configuration Error Interrupt
LL_DMA2D_IT_CTCIE	CLUT Transfer Complete Interrupt
LL_DMA2D_IT_CAEIE	CLUT Access Error Interrupt
LL_DMA2D_IT_TWIE	Transfer Watermark Interrupt
LL_DMA2D_IT_TCIE	Transfer Complete Interrupt

LL_DMA2D_IT_TEIE	Transfer Error Interrupt

Line Offset Mode

LL_DMA2D_LINE_OFFSET_PIXELS	Line offsets are expressed in pixels
LL_DMA2D_LINE_OFFSET_BYTES	Line offsets are expressed in bytes

Mode

LL_DMA2D_MODE_M2M	DMA2D memory to memory transfer mode
LL_DMA2D_MODE_M2M_PFC	DMA2D memory to memory with pixel format conversion transfer mode
LL_DMA2D_MODE_M2M_BLEND	DMA2D memory to memory with blending transfer mode
LL_DMA2D_MODE_R2M	DMA2D register to memory transfer mode
LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG	DMA2D memory to memory with blending transfer mode and fixed color foreground
LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG	DMA2D memory to memory with blending transfer mode and fixed color background

Output Color Mode

LL_DMA2D_OUTPUT_MODE_ARGB8888	ARGB8888
LL_DMA2D_OUTPUT_MODE_RGB888	RGB888
LL_DMA2D_OUTPUT_MODE_RGB565	RGB565
LL_DMA2D_OUTPUT_MODE_ARGB1555	ARGB1555
LL_DMA2D_OUTPUT_MODE_ARGB4444	ARGB4444

Swap Mode

LL_DMA2D_SWAP_MODE_REGULAR	Regular order
LL_DMA2D_SWAP_MODE_TWO_BY_TWO	Bytes swapped two by two

Red Blue Swap

LL_DMA2D_RB_MODE_REGULAR	RGB or ARGB
LL_DMA2D_RB_MODE_SWAP	BGR or ABGR

Common Write and read registers Macros

LL_DMA2D_WriteReg

Description:

- Write a value in DMA2D register.

Parameters:

- __INSTANCE__: DMA2D Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMA2D_ReadReg

Description:

- Read a value in DMA2D register.

Parameters:

- __INSTANCE__: DMA2D Instance
- __REG__: Register to be read

Return value:

- Register: value

81 LL DMAMUX Generic Driver

81.1 DMAMUX Firmware driver API description

81.1.1 Detailed description of functions

LL_DMAMUX_SetRequestID

Function name	**__STATIC_INLINE void LL_DMAMUX_SetRequestID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t Request)**
Function description	Set DMAMUX request ID for DMAMUX Channel x.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13 • **Request:** This parameter can be one of the following values: – LL_DMAMUX_REQ_MEM2MEM – LL_DMAMUX_REQ_GENERATOR0 – LL_DMAMUX_REQ_GENERATOR1 – LL_DMAMUX_REQ_GENERATOR2 – LL_DMAMUX_REQ_GENERATOR3 – LL_DMAMUX_REQ_ADC1 – LL_DMAMUX_REQ_DAC1_CH1 – LL_DMAMUX_REQ_DAC1_CH2 – LL_DMAMUX_REQ_TIM6_UP – LL_DMAMUX_REQ_TIM7_UP – LL_DMAMUX_REQ_SPI1_RX – LL_DMAMUX_REQ_SPI1_TX – LL_DMAMUX_REQ_SPI2_RX – LL_DMAMUX_REQ_SPI2_TX – LL_DMAMUX_REQ_SPI3_RX – LL_DMAMUX_REQ_SPI3_TX – LL_DMAMUX_REQ_I2C1_RX – LL_DMAMUX_REQ_I2C1_TX – LL_DMAMUX_REQ_I2C2_RX – LL_DMAMUX_REQ_I2C2_TX

- LL_DMAMUX_REQ_I2C3_RX
- LL_DMAMUX_REQ_I2C3_TX
- LL_DMAMUX_REQ_I2C4_RX
- LL_DMAMUX_REQ_I2C4_TX
- LL_DMAMUX_REQ_USART1_RX
- LL_DMAMUX_REQ_USART1_TX
- LL_DMAMUX_REQ_USART2_RX
- LL_DMAMUX_REQ_USART2_TX
- LL_DMAMUX_REQ_USART3_RX
- LL_DMAMUX_REQ_USART3_TX
- LL_DMAMUX_REQ_UART4_RX
- LL_DMAMUX_REQ_UART4_TX
- LL_DMAMUX_REQ_UART5_RX
- LL_DMAMUX_REQ_UART5_TX
- LL_DMAMUX_REQ_LPUART1_RX
- LL_DMAMUX_REQ_LPUART1_TX
- LL_DMAMUX_REQ_SAI1_A
- LL_DMAMUX_REQ_SAI1_B
- LL_DMAMUX_REQ_SAI2_A
- LL_DMAMUX_REQ_SAI2_B
- LL_DMAMUX_REQ_OSPI1
- LL_DMAMUX_REQ_OSPI2
- LL_DMAMUX_REQ_TIM1_CH1
- LL_DMAMUX_REQ_TIM1_CH2
- LL_DMAMUX_REQ_TIM1_CH3
- LL_DMAMUX_REQ_TIM1_CH4
- LL_DMAMUX_REQ_TIM1_UP
- LL_DMAMUX_REQ_TIM1_TRIG
- LL_DMAMUX_REQ_TIM1_COM
- LL_DMAMUX_REQ_TIM8_CH1
- LL_DMAMUX_REQ_TIM8_CH2
- LL_DMAMUX_REQ_TIM8_CH3
- LL_DMAMUX_REQ_TIM8_CH4
- LL_DMAMUX_REQ_TIM8_UP
- LL_DMAMUX_REQ_TIM8_TRIG
- LL_DMAMUX_REQ_TIM8_COM
- LL_DMAMUX_REQ_TIM2_CH1
- LL_DMAMUX_REQ_TIM2_CH2
- LL_DMAMUX_REQ_TIM2_CH3
- LL_DMAMUX_REQ_TIM2_CH4
- LL_DMAMUX_REQ_TIM2_UP
- LL_DMAMUX_REQ_TIM3_CH1
- LL_DMAMUX_REQ_TIM3_CH2
- LL_DMAMUX_REQ_TIM3_CH3
- LL_DMAMUX_REQ_TIM3_CH4
- LL_DMAMUX_REQ_TIM3_UP
- LL_DMAMUX_REQ_TIM3_TRIG
- LL_DMAMUX_REQ_TIM4_CH1
- LL_DMAMUX_REQ_TIM4_CH2
- LL_DMAMUX_REQ_TIM4_CH3
- LL_DMAMUX_REQ_TIM4_CH4
- LL_DMAMUX_REQ_TIM4_UP

- LL_DMAMUX_REQ_TIM5_CH1
- LL_DMAMUX_REQ_TIM5_CH2
- LL_DMAMUX_REQ_TIM5_CH3
- LL_DMAMUX_REQ_TIM5_CH4
- LL_DMAMUX_REQ_TIM5_UP
- LL_DMAMUX_REQ_TIM5_TRIG
- LL_DMAMUX_REQ_TIM15_CH1
- LL_DMAMUX_REQ_TIM15_UP
- LL_DMAMUX_REQ_TIM15_TRIG
- LL_DMAMUX_REQ_TIM15_COM
- LL_DMAMUX_REQ_TIM16_CH1
- LL_DMAMUX_REQ_TIM16_UP
- LL_DMAMUX_REQ_TIM17_CH1
- LL_DMAMUX_REQ_TIM17_UP
- LL_DMAMUX_REQ_DFSDM1_FLT0
- LL_DMAMUX_REQ_DFSDM1_FLT1
- LL_DMAMUX_REQ_DFSDM1_FLT2
- LL_DMAMUX_REQ_DFSDM1_FLT3
- LL_DMAMUX_REQ_DCMI
- LL_DMAMUX_REQ_AES_IN
- LL_DMAMUX_REQ_AES_OUT
- LL_DMAMUX_REQ_HASH_IN

Return values
- **None:**

Notes
- DMAMUX channel 0 to 6 are mapped to DMA1 channel 1 to 7. DMAMUX channel 7 to 13 are mapped to DMA2 channel 1 to 7.

Reference Manual to LL API cross reference:
- CxCR DMAREQ_ID LL_DMAMUX_SetRequestID

LL_DMAMUX_GetRequestID

Function name
__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description
Get DMAMUX request ID for DMAMUX Channel x.

Parameters
- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12

- LL_DMAMUX_CHANNEL_13

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_MEM2MEM
  - LL_DMAMUX_REQ_GENERATOR0
  - LL_DMAMUX_REQ_GENERATOR1
  - LL_DMAMUX_REQ_GENERATOR2
  - LL_DMAMUX_REQ_GENERATOR3
  - LL_DMAMUX_REQ_ADC1
  - LL_DMAMUX_REQ_DAC1_CH1
  - LL_DMAMUX_REQ_DAC1_CH2
  - LL_DMAMUX_REQ_TIM6_UP
  - LL_DMAMUX_REQ_TIM7_UP
  - LL_DMAMUX_REQ_SPI1_RX
  - LL_DMAMUX_REQ_SPI1_TX
  - LL_DMAMUX_REQ_SPI2_RX
  - LL_DMAMUX_REQ_SPI2_TX
  - LL_DMAMUX_REQ_SPI3_RX
  - LL_DMAMUX_REQ_SPI3_TX
  - LL_DMAMUX_REQ_I2C1_RX
  - LL_DMAMUX_REQ_I2C1_TX
  - LL_DMAMUX_REQ_I2C2_RX
  - LL_DMAMUX_REQ_I2C2_TX
  - LL_DMAMUX_REQ_I2C3_RX
  - LL_DMAMUX_REQ_I2C3_TX
  - LL_DMAMUX_REQ_I2C4_RX
  - LL_DMAMUX_REQ_I2C4_TX
  - LL_DMAMUX_REQ_USART1_RX
  - LL_DMAMUX_REQ_USART1_TX
  - LL_DMAMUX_REQ_USART2_RX
  - LL_DMAMUX_REQ_USART2_TX
  - LL_DMAMUX_REQ_USART3_RX
  - LL_DMAMUX_REQ_USART3_TX
  - LL_DMAMUX_REQ_UART4_RX
  - LL_DMAMUX_REQ_UART4_TX
  - LL_DMAMUX_REQ_UART5_RX
  - LL_DMAMUX_REQ_UART5_TX
  - LL_DMAMUX_REQ_LPUART1_RX
  - LL_DMAMUX_REQ_LPUART1_TX
  - LL_DMAMUX_REQ_SAI1_A
  - LL_DMAMUX_REQ_SAI1_B
  - LL_DMAMUX_REQ_SAI2_A
  - LL_DMAMUX_REQ_SAI2_B
  - LL_DMAMUX_REQ_OSPI1
  - LL_DMAMUX_REQ_OSPI2
  - LL_DMAMUX_REQ_TIM1_CH1
  - LL_DMAMUX_REQ_TIM1_CH2
  - LL_DMAMUX_REQ_TIM1_CH3
  - LL_DMAMUX_REQ_TIM1_CH4
  - LL_DMAMUX_REQ_TIM1_UP
  - LL_DMAMUX_REQ_TIM1_TRIG
  - LL_DMAMUX_REQ_TIM1_COM
  - LL_DMAMUX_REQ_TIM8_CH1

- LL_DMAMUX_REQ_TIM8_CH2
- LL_DMAMUX_REQ_TIM8_CH3
- LL_DMAMUX_REQ_TIM8_CH4
- LL_DMAMUX_REQ_TIM8_UP
- LL_DMAMUX_REQ_TIM8_TRIG
- LL_DMAMUX_REQ_TIM8_COM
- LL_DMAMUX_REQ_TIM2_CH1
- LL_DMAMUX_REQ_TIM2_CH2
- LL_DMAMUX_REQ_TIM2_CH3
- LL_DMAMUX_REQ_TIM2_CH4
- LL_DMAMUX_REQ_TIM2_UP
- LL_DMAMUX_REQ_TIM3_CH1
- LL_DMAMUX_REQ_TIM3_CH2
- LL_DMAMUX_REQ_TIM3_CH3
- LL_DMAMUX_REQ_TIM3_CH4
- LL_DMAMUX_REQ_TIM3_UP
- LL_DMAMUX_REQ_TIM3_TRIG
- LL_DMAMUX_REQ_TIM4_CH1
- LL_DMAMUX_REQ_TIM4_CH2
- LL_DMAMUX_REQ_TIM4_CH3
- LL_DMAMUX_REQ_TIM4_CH4
- LL_DMAMUX_REQ_TIM4_UP
- LL_DMAMUX_REQ_TIM5_CH1
- LL_DMAMUX_REQ_TIM5_CH2
- LL_DMAMUX_REQ_TIM5_CH3
- LL_DMAMUX_REQ_TIM5_CH4
- LL_DMAMUX_REQ_TIM5_UP
- LL_DMAMUX_REQ_TIM5_TRIG
- LL_DMAMUX_REQ_TIM15_CH1
- LL_DMAMUX_REQ_TIM15_UP
- LL_DMAMUX_REQ_TIM15_TRIG
- LL_DMAMUX_REQ_TIM15_COM
- LL_DMAMUX_REQ_TIM16_CH1
- LL_DMAMUX_REQ_TIM16_UP
- LL_DMAMUX_REQ_TIM17_CH1
- LL_DMAMUX_REQ_TIM17_UP
- LL_DMAMUX_REQ_DFSDM1_FLT0
- LL_DMAMUX_REQ_DFSDM1_FLT1
- LL_DMAMUX_REQ_DFSDM1_FLT2
- LL_DMAMUX_REQ_DFSDM1_FLT3
- LL_DMAMUX_REQ_DCMI
- LL_DMAMUX_REQ_AES_IN
- LL_DMAMUX_REQ_AES_OUT
- LL_DMAMUX_REQ_HASH_IN

Notes
- DMAMUX channel 0 to 6 are mapped to DMA1 channel 1 to 7. DMAMUX channel 7 to 13 are mapped to DMA2 channel 1 to 7.

Reference Manual to LL API cross reference:
- CxCR DMAREQ_ID LL_DMAMUX_GetRequestID

LL_DMAMUX_SetSyncRequestNb

Function name	**__STATIC_INLINE void LL_DMAMUX_SetSyncRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t RequestNb)**
Function description	Set the number of DMA request that will be autorized after a synchronization event and/or the number of DMA request needed to generate an event.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13 • **RequestNb:** This parameter must be a value between Min_Data = 1 and Max_Data = 32.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CxCR NBREQ LL_DMAMUX_SetSyncRequestNb

LL_DMAMUX_GetSyncRequestNb

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Get the number of DMA request that will be autorized after a synchronization event and/or the number of DMA request needed to generate an event.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10

	– LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13
Return values	• **Between:** Min_Data = 1 and Max_Data = 32
Reference Manual to LL API cross reference:	• CxCR NBREQ LL_DMAMUX_GetSyncRequestNb

LL_DMAMUX_SetSyncPolarity

Function name	**__STATIC_INLINE void LL_DMAMUX_SetSyncPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t Polarity)**
Function description	Set the polarity of the signal on which the DMA request is synchronized.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13 • **Polarity:** This parameter can be one of the following values: – LL_DMAMUX_SYNC_NO_EVENT – LL_DMAMUX_SYNC_POL_RISING – LL_DMAMUX_SYNC_POL_FALLING – LL_DMAMUX_SYNC_POL_RISING_FALLING
Return values	• **None:**
Reference Manual to LL API cross reference:	• CxCR SPOL LL_DMAMUX_SetSyncPolarity

LL_DMAMUX_GetSyncPolarity

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Get the polarity of the signal on which the DMA request is synchronized.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values:

- LL_DMAMUX_CHANNEL_0
- LL_DMAMUX_CHANNEL_1
- LL_DMAMUX_CHANNEL_2
- LL_DMAMUX_CHANNEL_3
- LL_DMAMUX_CHANNEL_4
- LL_DMAMUX_CHANNEL_5
- LL_DMAMUX_CHANNEL_6
- LL_DMAMUX_CHANNEL_7
- LL_DMAMUX_CHANNEL_8
- LL_DMAMUX_CHANNEL_9
- LL_DMAMUX_CHANNEL_10
- LL_DMAMUX_CHANNEL_11
- LL_DMAMUX_CHANNEL_12
- LL_DMAMUX_CHANNEL_13

Return values
- **Returned:** value can be one of the following values:
  - LL_DMAMUX_SYNC_NO_EVENT
  - LL_DMAMUX_SYNC_POL_RISING
  - LL_DMAMUX_SYNC_POL_FALLING
  - LL_DMAMUX_SYNC_POL_RISING_FALLING

Reference Manual to LL API cross reference:
- CxCR SPOL LL_DMAMUX_GetSyncPolarity

LL_DMAMUX_EnableEventGeneration

Function name
__STATIC_INLINE void LL_DMAMUX_EnableEventGeneration (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description
Enable the Event Generation on DMAMUX channel x.

Parameters
- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13

Return values
- **None:**

Reference Manual to LL API cross reference:
- CxCR EGE LL_DMAMUX_EnableEventGeneration

LL_DMAMUX_DisableEventGeneration

Function name	**__STATIC_INLINE void LL_DMAMUX_DisableEventGeneration (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Disable the Event Generation on DMAMUX channel x.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13
Return values	• **None:**
Reference Manual to LL API cross reference:	• CxCR EGE LL_DMAMUX_DisableEventGeneration

LL_DMAMUX_IsEnabledEventGeneration

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledEventGeneration (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Check if the Event Generation on DMAMUX channel x is enabled or disabled.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CxCR EGE LL_DMAMUX_IsEnabledEventGeneration

LL_DMAMUX_EnableSync

Function name	**__STATIC_INLINE void LL_DMAMUX_EnableSync (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Enable the synchronization mode.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13
Return values	• **None:**
Reference Manual to LL API cross reference:	• CxCR SE LL_DMAMUX_EnableSync

LL_DMAMUX_DisableSync

Function name	**__STATIC_INLINE void LL_DMAMUX_DisableSync (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Disable the synchronization mode.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11

– LL_DMAMUX_CHANNEL_12
– LL_DMAMUX_CHANNEL_13

Return values	• **None:**
Reference Manual to LL API cross reference:	• CxCR SE LL_DMAMUX_DisableSync

LL_DMAMUX_IsEnabledSync

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledSync (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Check if the synchronization mode is enabled or disabled.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CxCR SE LL_DMAMUX_IsEnabledSync

LL_DMAMUX_SetSyncID

Function name	**__STATIC_INLINE void LL_DMAMUX_SetSyncID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t SyncID)**
Function description	Set DMAMUX synchronization ID on DMAMUX Channel x.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7

- LL_DMAMUX_CHANNEL_8
- LL_DMAMUX_CHANNEL_9
- LL_DMAMUX_CHANNEL_10
- LL_DMAMUX_CHANNEL_11
- LL_DMAMUX_CHANNEL_12
- LL_DMAMUX_CHANNEL_13

	• **SyncID:** This parameter can be one of the following values: – LL_DMAMUX_SYNC_EXTI_LINE0 – LL_DMAMUX_SYNC_EXTI_LINE1 – LL_DMAMUX_SYNC_EXTI_LINE2 – LL_DMAMUX_SYNC_EXTI_LINE3 – LL_DMAMUX_SYNC_EXTI_LINE4 – LL_DMAMUX_SYNC_EXTI_LINE5 – LL_DMAMUX_SYNC_EXTI_LINE6 – LL_DMAMUX_SYNC_EXTI_LINE7 – LL_DMAMUX_SYNC_EXTI_LINE8 – LL_DMAMUX_SYNC_EXTI_LINE9 – LL_DMAMUX_SYNC_EXTI_LINE10 – LL_DMAMUX_SYNC_EXTI_LINE11 – LL_DMAMUX_SYNC_EXTI_LINE12 – LL_DMAMUX_SYNC_EXTI_LINE13 – LL_DMAMUX_SYNC_EXTI_LINE14 – LL_DMAMUX_SYNC_EXTI_LINE15 – LL_DMAMUX_SYNC_DMAMUX_CH0 – LL_DMAMUX_SYNC_DMAMUX_CH1 – LL_DMAMUX_SYNC_DMAMUX_CH2 – LL_DMAMUX_SYNC_DMAMUX_CH3 – LL_DMAMUX_SYNC_LPTIM1_OUT – LL_DMAMUX_SYNC_LPTIM2_OUT – LL_DMAMUX_SYNC_DSI_TE – LL_DMAMUX_SYNC_DSI_REFRESH_END – LL_DMAMUX_SYNC_DMA2D_TX_END – LL_DMAMUX_SYNC_LTDC_LINE_IT
Return values	• **None:**
Reference Manual to LL API cross reference:	• CxCR SYNC_ID LL_DMAMUX_SetSyncID

LL_DMAMUX_GetSyncID

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Get DMAMUX synchronization ID on DMAMUX Channel x.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5

	– LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13
Return values	• **Returned:** value can be one of the following values: – LL_DMAMUX_SYNC_EXTI_LINE0 – LL_DMAMUX_SYNC_EXTI_LINE1 – LL_DMAMUX_SYNC_EXTI_LINE2 – LL_DMAMUX_SYNC_EXTI_LINE3 – LL_DMAMUX_SYNC_EXTI_LINE4 – LL_DMAMUX_SYNC_EXTI_LINE5 – LL_DMAMUX_SYNC_EXTI_LINE6 – LL_DMAMUX_SYNC_EXTI_LINE7 – LL_DMAMUX_SYNC_EXTI_LINE8 – LL_DMAMUX_SYNC_EXTI_LINE9 – LL_DMAMUX_SYNC_EXTI_LINE10 – LL_DMAMUX_SYNC_EXTI_LINE11 – LL_DMAMUX_SYNC_EXTI_LINE12 – LL_DMAMUX_SYNC_EXTI_LINE13 – LL_DMAMUX_SYNC_EXTI_LINE14 – LL_DMAMUX_SYNC_EXTI_LINE15 – LL_DMAMUX_SYNC_DMAMUX_CH0 – LL_DMAMUX_SYNC_DMAMUX_CH1 – LL_DMAMUX_SYNC_DMAMUX_CH2 – LL_DMAMUX_SYNC_DMAMUX_CH3 – LL_DMAMUX_SYNC_LPTIM1_OUT – LL_DMAMUX_SYNC_LPTIM2_OUT – LL_DMAMUX_SYNC_DSI_TE – LL_DMAMUX_SYNC_DSI_REFRESH_END – LL_DMAMUX_SYNC_DMA2D_TX_END – LL_DMAMUX_SYNC_LTDC_LINE_IT
Reference Manual to LL API cross reference:	• CxCR SYNC_ID LL_DMAMUX_GetSyncID

LL_DMAMUX_EnableRequestGen

Function name	**__STATIC_INLINE void LL_DMAMUX_EnableRequestGen (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)**
Function description	Enable the Request Generator.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2

– LL_DMAMUX_REQ_GEN_3

Return values	• **None:**
Reference Manual to LL API cross reference:	• RGxCR GE LL_DMAMUX_EnableRequestGen

LL_DMAMUX_DisableRequestGen

Function name	**__STATIC_INLINE void LL_DMAMUX_DisableRequestGen (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)**
Function description	Disable the Request Generator.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2 – LL_DMAMUX_REQ_GEN_3
Return values	• **None:**
Reference Manual to LL API cross reference:	• RGxCR GE LL_DMAMUX_DisableRequestGen

LL_DMAMUX_IsEnabledRequestGen

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledRequestGen (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)**
Function description	Check if the Request Generator is enabled or disabled.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2 – LL_DMAMUX_REQ_GEN_3
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• RGxCR GE LL_DMAMUX_IsEnabledRequestGen

LL_DMAMUX_SetRequestGenPolarity

Function name	**__STATIC_INLINE void LL_DMAMUX_SetRequestGenPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t Polarity)**

Function description	Set the polarity of the signal on which the DMA request is generated.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2 – LL_DMAMUX_REQ_GEN_3 • **Polarity:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_NO_EVENT – LL_DMAMUX_REQ_GEN_POL_RISING – LL_DMAMUX_REQ_GEN_POL_FALLING – LL_DMAMUX_REQ_GEN_POL_RISING_FALLING
Return values	• **None:**
Reference Manual to LL API cross reference:	• RGxCR GPOL LL_DMAMUX_SetRequestGenPolarity

LL_DMAMUX_GetRequestGenPolarity

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestGenPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)**
Function description	Get the polarity of the signal on which the DMA request is generated.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2 – LL_DMAMUX_REQ_GEN_3
Return values	• **Returned:** value can be one of the following values: – LL_DMAMUX_REQ_GEN_NO_EVENT – LL_DMAMUX_REQ_GEN_POL_RISING – LL_DMAMUX_REQ_GEN_POL_FALLING – LL_DMAMUX_REQ_GEN_POL_RISING_FALLING
Reference Manual to LL API cross reference:	• RGxCR GPOL LL_DMAMUX_GetRequestGenPolarity

LL_DMAMUX_SetGenRequestNb

Function name	**__STATIC_INLINE void LL_DMAMUX_SetGenRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t RequestNb)**
Function description	Set the number of DMA request that will be autorized after a

	generation event.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2 – LL_DMAMUX_REQ_GEN_3 • **RequestNb:** This parameter must be a value between Min_Data = 1 and Max_Data = 32.
Return values	• **None:**
Notes	• This field can only be written when Generator is disabled.
Reference Manual to LL API cross reference:	• RGxCR GNBREQ LL_DMAMUX_SetGenRequestNb

LL_DMAMUX_GetGenRequestNb

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_GetGenRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)**
Function description	Get the number of DMA request that will be autorized after a generation event.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2 – LL_DMAMUX_REQ_GEN_3
Return values	• **Between:** Min_Data = 1 and Max_Data = 32
Reference Manual to LL API cross reference:	• RGxCR GNBREQ LL_DMAMUX_GetGenRequestNb

LL_DMAMUX_SetRequestSignalID

Function name	**__STATIC_INLINE void LL_DMAMUX_SetRequestSignalID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t RequestSignalID)**
Function description	Set DMAMUX external Request Signal ID on DMAMUX Request Generation Trigger Event Channel x.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2 – LL_DMAMUX_REQ_GEN_3

- **RequestSignalID:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_EXTI_LINE0
  - LL_DMAMUX_REQ_GEN_EXTI_LINE1
  - LL_DMAMUX_REQ_GEN_EXTI_LINE2
  - LL_DMAMUX_REQ_GEN_EXTI_LINE3
  - LL_DMAMUX_REQ_GEN_EXTI_LINE4
  - LL_DMAMUX_REQ_GEN_EXTI_LINE5
  - LL_DMAMUX_REQ_GEN_EXTI_LINE6
  - LL_DMAMUX_REQ_GEN_EXTI_LINE7
  - LL_DMAMUX_REQ_GEN_EXTI_LINE8
  - LL_DMAMUX_REQ_GEN_EXTI_LINE9
  - LL_DMAMUX_REQ_GEN_EXTI_LINE10
  - LL_DMAMUX_REQ_GEN_EXTI_LINE11
  - LL_DMAMUX_REQ_GEN_EXTI_LINE12
  - LL_DMAMUX_REQ_GEN_EXTI_LINE13
  - LL_DMAMUX_REQ_GEN_EXTI_LINE14
  - LL_DMAMUX_REQ_GEN_EXTI_LINE15
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH0
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH1
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH2
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH3
  - LL_DMAMUX_REQ_GEN_LPTIM1_OUT
  - LL_DMAMUX_REQ_GEN_LPTIM2_OUT
  - LL_DMAMUX_REQ_GEN_DSI_TE
  - LL_DMAMUX_REQ_GEN_DSI_REFRESH_END
  - LL_DMAMUX_REQ_GEN_DMA2D_TX_END
  - LL_DMAMUX_REQ_GEN_LTDC_LINE_IT

Return values
- **None:**

Reference Manual to LL API cross reference:
- RGxCR SIG_ID LL_DMAMUX_SetRequestSignalID

LL_DMAMUX_GetRequestSignalID

Function name
__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestSignalID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description
Get DMAMUX external Request Signal ID set on DMAMUX Channel x.

Parameters
- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values
- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_GEN_EXTI_LINE0
  - LL_DMAMUX_REQ_GEN_EXTI_LINE1

- LL_DMAMUX_REQ_GEN_EXTI_LINE2
- LL_DMAMUX_REQ_GEN_EXTI_LINE3
- LL_DMAMUX_REQ_GEN_EXTI_LINE4
- LL_DMAMUX_REQ_GEN_EXTI_LINE5
- LL_DMAMUX_REQ_GEN_EXTI_LINE6
- LL_DMAMUX_REQ_GEN_EXTI_LINE7
- LL_DMAMUX_REQ_GEN_EXTI_LINE8
- LL_DMAMUX_REQ_GEN_EXTI_LINE9
- LL_DMAMUX_REQ_GEN_EXTI_LINE10
- LL_DMAMUX_REQ_GEN_EXTI_LINE11
- LL_DMAMUX_REQ_GEN_EXTI_LINE12
- LL_DMAMUX_REQ_GEN_EXTI_LINE13
- LL_DMAMUX_REQ_GEN_EXTI_LINE14
- LL_DMAMUX_REQ_GEN_EXTI_LINE15
- LL_DMAMUX_REQ_GEN_DMAMUX_CH0
- LL_DMAMUX_REQ_GEN_DMAMUX_CH1
- LL_DMAMUX_REQ_GEN_DMAMUX_CH2
- LL_DMAMUX_REQ_GEN_DMAMUX_CH3
- LL_DMAMUX_REQ_GEN_LPTIM1_OUT
- LL_DMAMUX_REQ_GEN_LPTIM2_OUT
- LL_DMAMUX_REQ_GEN_DSI_TE
- LL_DMAMUX_REQ_GEN_DSI_REFRESH_END
- LL_DMAMUX_REQ_GEN_DMA2D_TX_END
- LL_DMAMUX_REQ_GEN_LTDC_LINE_IT

Reference Manual to LL API cross reference:	• RGxCR SIG_ID LL_DMAMUX_GetRequestSignalID

LL_DMAMUX_IsActiveFlag_SO0

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO0 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 0.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF0 LL_DMAMUX_IsActiveFlag_SO0

LL_DMAMUX_IsActiveFlag_SO1

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO1 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 1.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CSR SOF1 LL_DMAMUX_IsActiveFlag_SO1

reference:

LL_DMAMUX_IsActiveFlag_SO2

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO2 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 2.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF2 LL_DMAMUX_IsActiveFlag_SO2

LL_DMAMUX_IsActiveFlag_SO3

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO3 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 3.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF3 LL_DMAMUX_IsActiveFlag_SO3

LL_DMAMUX_IsActiveFlag_SO4

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO4 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 4.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF4 LL_DMAMUX_IsActiveFlag_SO4

LL_DMAMUX_IsActiveFlag_SO5

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO5 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 5.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF5 LL_DMAMUX_IsActiveFlag_SO5

LL_DMAMUX_IsActiveFlag_SO6

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO6 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 6.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF6 LL_DMAMUX_IsActiveFlag_SO6

LL_DMAMUX_IsActiveFlag_SO7

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO7 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 7.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF7 LL_DMAMUX_IsActiveFlag_SO7

LL_DMAMUX_IsActiveFlag_SO8

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO8 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 8.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF8 LL_DMAMUX_IsActiveFlag_SO8

LL_DMAMUX_IsActiveFlag_SO9

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO9 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 9.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF9 LL_DMAMUX_IsActiveFlag_SO9

LL_DMAMUX_IsActiveFlag_SO10

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO10 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 10.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF10 LL_DMAMUX_IsActiveFlag_SO10

LL_DMAMUX_IsActiveFlag_SO11

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO11 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 11.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF11 LL_DMAMUX_IsActiveFlag_SO11

LL_DMAMUX_IsActiveFlag_SO12

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO12 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 12.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF12 LL_DMAMUX_IsActiveFlag_SO12

LL_DMAMUX_IsActiveFlag_SO13

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO13 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Synchronization Event Overrun Flag Channel 13.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SOF13 LL_DMAMUX_IsActiveFlag_SO13

LL_DMAMUX_IsActiveFlag_RGO0

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO0 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Request Generator 0 Trigger Event Overrun Flag.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• RGSR OF0 LL_DMAMUX_IsActiveFlag_RGO0

LL_DMAMUX_IsActiveFlag_RGO1

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO1 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Request Generator 1 Trigger Event Overrun Flag.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• RGSR OF1 LL_DMAMUX_IsActiveFlag_RGO1

LL_DMAMUX_IsActiveFlag_RGO2

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO2 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Request Generator 2 Trigger Event Overrun Flag.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• RGSR OF2 LL_DMAMUX_IsActiveFlag_RGO2

LL_DMAMUX_IsActiveFlag_RGO3

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO3 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Get Request Generator 3 Trigger Event Overrun Flag.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• RGSR OF3 LL_DMAMUX_IsActiveFlag_RGO3

LL_DMAMUX_ClearFlag_SO0

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO0 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 0.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF0 LL_DMAMUX_ClearFlag_SO0

LL_DMAMUX_ClearFlag_SO1

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO1 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 1.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF1 LL_DMAMUX_ClearFlag_SO1

LL_DMAMUX_ClearFlag_SO2

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO2 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 2.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF2 LL_DMAMUX_ClearFlag_SO2

LL_DMAMUX_ClearFlag_SO3

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO3 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 3.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF3 LL_DMAMUX_ClearFlag_SO3

LL_DMAMUX_ClearFlag_SO4

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO4 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 4.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF4 LL_DMAMUX_ClearFlag_SO4

LL_DMAMUX_ClearFlag_SO5

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO5 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 5.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF5 LL_DMAMUX_ClearFlag_SO5

LL_DMAMUX_ClearFlag_SO6

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO6 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 6.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF6 LL_DMAMUX_ClearFlag_SO6

LL_DMAMUX_ClearFlag_SO7

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO7 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 7.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF7 LL_DMAMUX_ClearFlag_SO7

LL_DMAMUX_ClearFlag_SO8

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO8 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 8.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF8 LL_DMAMUX_ClearFlag_SO8

LL_DMAMUX_ClearFlag_SO9

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO9 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 9.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF9 LL_DMAMUX_ClearFlag_SO9

LL_DMAMUX_ClearFlag_SO10

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO10 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 10.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF10 LL_DMAMUX_ClearFlag_SO10

LL_DMAMUX_ClearFlag_SO11

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO11 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 11.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF11 LL_DMAMUX_ClearFlag_SO11

LL_DMAMUX_ClearFlag_SO12

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO12 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 12.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF12 LL_DMAMUX_ClearFlag_SO12

LL_DMAMUX_ClearFlag_SO13

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO13 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Synchronization Event Overrun Flag Channel 13.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFR CSOF13 LL_DMAMUX_ClearFlag_SO13

LL_DMAMUX_ClearFlag_RGO0

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO0 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Request Generator 0 Trigger Event Overrun Flag.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• RGCFR COF0 LL_DMAMUX_ClearFlag_RGO0

LL_DMAMUX_ClearFlag_RGO1

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO1 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Request Generator 1 Trigger Event Overrun Flag.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• RGCFR COF1 LL_DMAMUX_ClearFlag_RGO1

LL_DMAMUX_ClearFlag_RGO2

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO2 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Request Generator 2 Trigger Event Overrun Flag.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• RGCFR COF2 LL_DMAMUX_ClearFlag_RGO2

LL_DMAMUX_ClearFlag_RGO3

Function name	**__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO3 (DMAMUX_Channel_TypeDef * DMAMUXx)**
Function description	Clear Request Generator 3 Trigger Event Overrun Flag.
Parameters	• **DMAMUXx:** DMAMUXx DMAMUXx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• RGCFR COF3 LL_DMAMUX_ClearFlag_RGO3

LL_DMAMUX_EnableIT_SO

Function name	**__STATIC_INLINE void LL_DMAMUX_EnableIT_SO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Enable the Synchronization Event Overrun Interrupt on DMAMUX channel x.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13
Return values	• **None:**
Reference Manual to LL API cross	• CxCR SOIE LL_DMAMUX_EnableIT_SO

reference:

LL_DMAMUX_DisableIT_SO

Function name	**__STATIC_INLINE void LL_DMAMUX_DisableIT_SO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Disable the Synchronization Event Overrun Interrupt on DMAMUX channel x.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12 – LL_DMAMUX_CHANNEL_13
Return values	• **None:**
Reference Manual to LL API cross reference:	• CxCR SOIE LL_DMAMUX_DisableIT_SO

LL_DMAMUX_IsEnabledIT_SO

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledIT_SO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)**
Function description	Check if the Synchronization Event Overrun Interrupt on DMAMUX channel x is enabled or disabled.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **Channel:** This parameter can be one of the following values: – LL_DMAMUX_CHANNEL_0 – LL_DMAMUX_CHANNEL_1 – LL_DMAMUX_CHANNEL_2 – LL_DMAMUX_CHANNEL_3 – LL_DMAMUX_CHANNEL_4 – LL_DMAMUX_CHANNEL_5 – LL_DMAMUX_CHANNEL_6 – LL_DMAMUX_CHANNEL_7 – LL_DMAMUX_CHANNEL_8 – LL_DMAMUX_CHANNEL_9 – LL_DMAMUX_CHANNEL_10 – LL_DMAMUX_CHANNEL_11 – LL_DMAMUX_CHANNEL_12

– LL_DMAMUX_CHANNEL_13

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CxCR SOIE LL_DMAMUX_IsEnabledIT_SO

LL_DMAMUX_EnableIT_RGO

Function name	**__STATIC_INLINE void LL_DMAMUX_EnableIT_RGO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)**
Function description	Enable the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2 – LL_DMAMUX_REQ_GEN_3
Return values	• **None:**
Reference Manual to LL API cross reference:	• RGxCR OIE LL_DMAMUX_EnableIT_RGO

LL_DMAMUX_DisableIT_RGO

Function name	**__STATIC_INLINE void LL_DMAMUX_DisableIT_RGO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)**
Function description	Disable the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2 – LL_DMAMUX_REQ_GEN_3
Return values	• **None:**
Reference Manual to LL API cross reference:	• RGxCR OIE LL_DMAMUX_DisableIT_RGO

LL_DMAMUX_IsEnabledIT_RGO

Function name	**__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledIT_RGO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t**

	RequestGenChannel)
Function description	Check if the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x is enabled or disabled.
Parameters	• **DMAMUXx:** DMAMUXx Instance • **RequestGenChannel:** This parameter can be one of the following values: – LL_DMAMUX_REQ_GEN_0 – LL_DMAMUX_REQ_GEN_1 – LL_DMAMUX_REQ_GEN_2 – LL_DMAMUX_REQ_GEN_3
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• RGxCR OIE LL_DMAMUX_IsEnabledIT_RGO

81.2 DMAMUX Firmware driver defines

81.2.1 DMAMUX

DMAMUX Channel

LL_DMAMUX_CHANNEL_0	DMAMUX Channel 0 connected to DMA1 Channel 1
LL_DMAMUX_CHANNEL_1	DMAMUX Channel 1 connected to DMA1 Channel 2
LL_DMAMUX_CHANNEL_2	DMAMUX Channel 2 connected to DMA1 Channel 3
LL_DMAMUX_CHANNEL_3	DMAMUX Channel 3 connected to DMA1 Channel 4
LL_DMAMUX_CHANNEL_4	DMAMUX Channel 4 connected to DMA1 Channel 5
LL_DMAMUX_CHANNEL_5	DMAMUX Channel 5 connected to DMA1 Channel 6
LL_DMAMUX_CHANNEL_6	DMAMUX Channel 6 connected to DMA1 Channel 7
LL_DMAMUX_CHANNEL_7	DMAMUX Channel 7 connected to DMA2 Channel 1
LL_DMAMUX_CHANNEL_8	DMAMUX Channel 8 connected to DMA2 Channel 2
LL_DMAMUX_CHANNEL_9	DMAMUX Channel 9 connected to DMA2 Channel 3
LL_DMAMUX_CHANNEL_10	DMAMUX Channel 10 connected to DMA2 Channel 4
LL_DMAMUX_CHANNEL_11	DMAMUX Channel 11 connected to DMA2 Channel 5
LL_DMAMUX_CHANNEL_12	DMAMUX Channel 12 connected to DMA2 Channel 6
LL_DMAMUX_CHANNEL_13	DMAMUX Channel 13 connected to DMA2 Channel 7

Clear Flags Defines

LL_DMAMUX_CFR_CSOF0	Synchronization Event Overrun Flag Channel 0
LL_DMAMUX_CFR_CSOF1	Synchronization Event Overrun Flag Channel 1
LL_DMAMUX_CFR_CSOF2	Synchronization Event Overrun Flag Channel 2
LL_DMAMUX_CFR_CSOF3	Synchronization Event Overrun Flag Channel 3
LL_DMAMUX_CFR_CSOF4	Synchronization Event Overrun Flag Channel 4
LL_DMAMUX_CFR_CSOF5	Synchronization Event Overrun Flag Channel 5

LL_DMAMUX_CFR_CSOF6	Synchronization Event Overrun Flag Channel 6
LL_DMAMUX_CFR_CSOF7	Synchronization Event Overrun Flag Channel 7
LL_DMAMUX_CFR_CSOF8	Synchronization Event Overrun Flag Channel 8
LL_DMAMUX_CFR_CSOF9	Synchronization Event Overrun Flag Channel 9
LL_DMAMUX_CFR_CSOF10	Synchronization Event Overrun Flag Channel 10
LL_DMAMUX_CFR_CSOF11	Synchronization Event Overrun Flag Channel 11
LL_DMAMUX_CFR_CSOF12	Synchronization Event Overrun Flag Channel 12
LL_DMAMUX_CFR_CSOF13	Synchronization Event Overrun Flag Channel 13
LL_DMAMUX_RGCFR_RGCOF0	Request Generator 0 Trigger Event Overrun Flag
LL_DMAMUX_RGCFR_RGCOF1	Request Generator 1 Trigger Event Overrun Flag
LL_DMAMUX_RGCFR_RGCOF2	Request Generator 2 Trigger Event Overrun Flag
LL_DMAMUX_RGCFR_RGCOF3	Request Generator 3 Trigger Event Overrun Flag

Get Flags Defines

LL_DMAMUX_CSR_SOF0	Synchronization Event Overrun Flag Channel 0
LL_DMAMUX_CSR_SOF1	Synchronization Event Overrun Flag Channel 1
LL_DMAMUX_CSR_SOF2	Synchronization Event Overrun Flag Channel 2
LL_DMAMUX_CSR_SOF3	Synchronization Event Overrun Flag Channel 3
LL_DMAMUX_CSR_SOF4	Synchronization Event Overrun Flag Channel 4
LL_DMAMUX_CSR_SOF5	Synchronization Event Overrun Flag Channel 5
LL_DMAMUX_CSR_SOF6	Synchronization Event Overrun Flag Channel 6
LL_DMAMUX_CSR_SOF7	Synchronization Event Overrun Flag Channel 7
LL_DMAMUX_CSR_SOF8	Synchronization Event Overrun Flag Channel 8
LL_DMAMUX_CSR_SOF9	Synchronization Event Overrun Flag Channel 9
LL_DMAMUX_CSR_SOF10	Synchronization Event Overrun Flag Channel 10
LL_DMAMUX_CSR_SOF11	Synchronization Event Overrun Flag Channel 11
LL_DMAMUX_CSR_SOF12	Synchronization Event Overrun Flag Channel 12
LL_DMAMUX_CSR_SOF13	Synchronization Event Overrun Flag Channel 13
LL_DMAMUX_RGSR_RGOF0	Request Generator 0 Trigger Event Overrun Flag
LL_DMAMUX_RGSR_RGOF1	Request Generator 1 Trigger Event Overrun Flag
LL_DMAMUX_RGSR_RGOF2	Request Generator 2 Trigger Event Overrun Flag
LL_DMAMUX_RGSR_RGOF3	Request Generator 3 Trigger Event Overrun Flag

IT Defines

LL_DMAMUX_CCR_SOIE	Synchronization Event Overrun Interrupt
LL_DMAMUX_RGCR_RGOIE	Request Generation Trigger Event Overrun Interrupt

Transfer request

LL_DMAMUX_REQUEST_MEM2MEM	Memory to memory transfer

LL_DMAMUX_REQUEST_GENERATOR0	DMAMUX request generator 0
LL_DMAMUX_REQUEST_GENERATOR1	DMAMUX request generator 1
LL_DMAMUX_REQUEST_GENERATOR2	DMAMUX request generator 2
LL_DMAMUX_REQUEST_GENERATOR3	DMAMUX request generator 3
LL_DMAMUX_REQUEST_ADC1	DMAMUX ADC1 request
LL_DMAMUX_REQUEST_DAC1_CH1	DMAMUX DAC1 CH1 request
LL_DMAMUX_REQUEST_DAC1_CH2	DMAMUX DAC1 CH2 request
LL_DMAMUX_REQUEST_TIM6_UP	DMAMUX TIM6 UP request
LL_DMAMUX_REQUEST_TIM7_UP	DMAMUX TIM7 UP request
LL_DMAMUX_REQUEST_SPI1_RX	DMAMUX SPI1 RX request
LL_DMAMUX_REQUEST_SPI1_TX	DMAMUX SPI1 TX request
LL_DMAMUX_REQUEST_SPI2_RX	DMAMUX SPI2 RX request
LL_DMAMUX_REQUEST_SPI2_TX	DMAMUX SPI2 TX request
LL_DMAMUX_REQUEST_SPI3_RX	DMAMUX SPI3 RX request
LL_DMAMUX_REQUEST_SPI3_TX	DMAMUX SPI3 TX request
LL_DMAMUX_REQUEST_I2C1_RX	DMAMUX I2C1 RX request
LL_DMAMUX_REQUEST_I2C1_TX	DMAMUX I2C1 TX request
LL_DMAMUX_REQUEST_I2C2_RX	DMAMUX I2C2 RX request
LL_DMAMUX_REQUEST_I2C2_TX	DMAMUX I2C2 TX request
LL_DMAMUX_REQUEST_I2C3_RX	DMAMUX I2C3 RX request
LL_DMAMUX_REQUEST_I2C3_TX	DMAMUX I2C3 TX request
LL_DMAMUX_REQUEST_I2C4_RX	DMAMUX I2C4 RX request
LL_DMAMUX_REQUEST_I2C4_TX	DMAMUX I2C4 TX request
LL_DMAMUX_REQUEST_USART1_RX	DMAMUX USART1 RX request
LL_DMAMUX_REQUEST_USART1_TX	DMAMUX USART1 TX request
LL_DMAMUX_REQUEST_USART2_RX	DMAMUX USART2 RX request
LL_DMAMUX_REQUEST_USART2_TX	DMAMUX USART2 TX request
LL_DMAMUX_REQUEST_USART3_RX	DMAMUX USART3 RX request
LL_DMAMUX_REQUEST_USART3_TX	DMAMUX USART3 TX request
LL_DMAMUX_REQUEST_UART4_RX	DMAMUX UART4 RX request
LL_DMAMUX_REQUEST_UART4_TX	DMAMUX UART4 TX request
LL_DMAMUX_REQUEST_UART5_RX	DMAMUX UART5 RX request
LL_DMAMUX_REQUEST_UART5_TX	DMAMUX UART5 TX request
LL_DMAMUX_REQUEST_LPUART1_RX	DMAMUX LPUART1 RX request
LL_DMAMUX_REQUEST_LPUART1_TX	DMAMUX LPUART1 TX request
LL_DMAMUX_REQUEST_SAI1_A	DMAMUX SAI1 A request

LL_DMAMUX_REQUEST_SAI1_B	DMAMUX SAI1 B request
LL_DMAMUX_REQUEST_SAI2_A	DMAMUX SAI2 A request
LL_DMAMUX_REQUEST_SAI2_B	DMAMUX SAI2 B request
LL_DMAMUX_REQUEST_OSPI1	DMAMUX OCTOSPI1 request
LL_DMAMUX_REQUEST_OSPI2	DMAMUX OCTOSPI2 request
LL_DMAMUX_REQUEST_TIM1_CH1	DMAMUX TIM1 CH1 request
LL_DMAMUX_REQUEST_TIM1_CH2	DMAMUX TIM1 CH2 request
LL_DMAMUX_REQUEST_TIM1_CH3	DMAMUX TIM1 CH3 request
LL_DMAMUX_REQUEST_TIM1_CH4	DMAMUX TIM1 CH4 request
LL_DMAMUX_REQUEST_TIM1_UP	DMAMUX TIM1 UP request
LL_DMAMUX_REQUEST_TIM1_TRIG	DMAMUX TIM1 TRIG request
LL_DMAMUX_REQUEST_TIM1_COM	DMAMUX TIM1 COM request
LL_DMAMUX_REQUEST_TIM8_CH1	DMAMUX TIM8 CH1 request
LL_DMAMUX_REQUEST_TIM8_CH2	DMAMUX TIM8 CH2 request
LL_DMAMUX_REQUEST_TIM8_CH3	DMAMUX TIM8 CH3 request
LL_DMAMUX_REQUEST_TIM8_CH4	DMAMUX TIM8 CH4 request
LL_DMAMUX_REQUEST_TIM8_UP	DMAMUX TIM8 UP request
LL_DMAMUX_REQUEST_TIM8_TRIG	DMAMUX TIM8 TRIG request
LL_DMAMUX_REQUEST_TIM8_COM	DMAMUX TIM8 COM request
LL_DMAMUX_REQUEST_TIM2_CH1	DMAMUX TIM2 CH1 request
LL_DMAMUX_REQUEST_TIM2_CH2	DMAMUX TIM2 CH2 request
LL_DMAMUX_REQUEST_TIM2_CH3	DMAMUX TIM2 CH3 request
LL_DMAMUX_REQUEST_TIM2_CH4	DMAMUX TIM2 CH4 request
LL_DMAMUX_REQUEST_TIM2_UP	DMAMUX TIM2 UP request
LL_DMAMUX_REQUEST_TIM3_CH1	DMAMUX TIM3 CH1 request
LL_DMAMUX_REQUEST_TIM3_CH2	DMAMUX TIM3 CH2 request
LL_DMAMUX_REQUEST_TIM3_CH3	DMAMUX TIM3 CH3 request
LL_DMAMUX_REQUEST_TIM3_CH4	DMAMUX TIM3 CH4 request
LL_DMAMUX_REQUEST_TIM3_UP	DMAMUX TIM3 UP request
LL_DMAMUX_REQUEST_TIM3_TRIG	DMAMUX TIM3 TRIG request
LL_DMAMUX_REQUEST_TIM4_CH1	DMAMUX TIM4 CH1 request
LL_DMAMUX_REQUEST_TIM4_CH2	DMAMUX TIM4 CH2 request
LL_DMAMUX_REQUEST_TIM4_CH3	DMAMUX TIM4 CH3 request
LL_DMAMUX_REQUEST_TIM4_CH4	DMAMUX TIM4 CH4 request
LL_DMAMUX_REQUEST_TIM4_UP	DMAMUX TIM4 UP request
LL_DMAMUX_REQUEST_TIM5_CH1	DMAMUX TIM5 CH1 request

LL_DMAMUX_REQUEST_TIM5_CH2	DMAMUX TIM5 CH2 request
LL_DMAMUX_REQUEST_TIM5_CH3	DMAMUX TIM5 CH3 request
LL_DMAMUX_REQUEST_TIM5_CH4	DMAMUX TIM5 CH4 request
LL_DMAMUX_REQUEST_TIM5_UP	DMAMUX TIM5 UP request
LL_DMAMUX_REQUEST_TIM5_TRIG	DMAMUX TIM5 TRIG request
LL_DMAMUX_REQUEST_TIM15_CH1	DMAMUX TIM15 CH1 request
LL_DMAMUX_REQUEST_TIM15_UP	DMAMUX TIM15 UP request
LL_DMAMUX_REQUEST_TIM15_TRIG	DMAMUX TIM15 TRIG request
LL_DMAMUX_REQUEST_TIM15_COM	DMAMUX TIM15 COM request
LL_DMAMUX_REQUEST_TIM16_CH1	DMAMUX TIM16 CH1 request
LL_DMAMUX_REQUEST_TIM16_UP	DMAMUX TIM16 UP request
LL_DMAMUX_REQUEST_TIM17_CH1	DMAMUX TIM17 CH1 request
LL_DMAMUX_REQUEST_TIM17_UP	DMAMUX TIM17 UP request
LL_DMAMUX_REQUEST_DFSDM1_FLT0	DMAMUX DFSDM1_FLT0 request
LL_DMAMUX_REQUEST_DFSDM1_FLT1	DMAMUX DFSDM1_FLT1 request
LL_DMAMUX_REQUEST_DFSDM1_FLT2	DMAMUX DFSDM1_FLT2 request
LL_DMAMUX_REQUEST_DFSDM1_FLT3	DMAMUX DFSDM1_FLT3 request
LL_DMAMUX_REQUEST_DCMI	DMAMUX DCMI request
LL_DMAMUX_REQUEST_AES_IN	DMAMUX AES_IN request
LL_DMAMUX_REQUEST_AES_OUT	DMAMUX AES_OUT request
LL_DMAMUX_REQUEST_HASH_IN	DMAMUX HASH_IN request
LL_DMAMUX_REQ_MEM2MEM	Memory to memory transfer
LL_DMAMUX_REQ_GENERATOR0	DMAMUX request generator 0
LL_DMAMUX_REQ_GENERATOR1	DMAMUX request generator 1
LL_DMAMUX_REQ_GENERATOR2	DMAMUX request generator 2
LL_DMAMUX_REQ_GENERATOR3	DMAMUX request generator 3
LL_DMAMUX_REQ_ADC1	DMAMUX ADC1 request
LL_DMAMUX_REQ_DAC1_CH1	DMAMUX DAC1 CH1 request
LL_DMAMUX_REQ_DAC1_CH2	DMAMUX DAC1 CH2 request
LL_DMAMUX_REQ_TIM6_UP	DMAMUX TIM6 UP request
LL_DMAMUX_REQ_TIM7_UP	DMAMUX TIM7 UP request
LL_DMAMUX_REQ_SPI1_RX	DMAMUX SPI1 RX request
LL_DMAMUX_REQ_SPI1_TX	DMAMUX SPI1 TX request
LL_DMAMUX_REQ_SPI2_RX	DMAMUX SPI2 RX request
LL_DMAMUX_REQ_SPI2_TX	DMAMUX SPI2 TX request
LL_DMAMUX_REQ_SPI3_RX	DMAMUX SPI3 RX request

LL_DMAMUX_REQ_SPI3_TX	DMAMUX SPI3 TX request
LL_DMAMUX_REQ_I2C1_RX	DMAMUX I2C1 RX request
LL_DMAMUX_REQ_I2C1_TX	DMAMUX I2C1 TX request
LL_DMAMUX_REQ_I2C2_RX	DMAMUX I2C2 RX request
LL_DMAMUX_REQ_I2C2_TX	DMAMUX I2C2 TX request
LL_DMAMUX_REQ_I2C3_RX	DMAMUX I2C3 RX request
LL_DMAMUX_REQ_I2C3_TX	DMAMUX I2C3 TX request
LL_DMAMUX_REQ_I2C4_RX	DMAMUX I2C4 RX request
LL_DMAMUX_REQ_I2C4_TX	DMAMUX I2C4 TX request
LL_DMAMUX_REQ_USART1_RX	DMAMUX USART1 RX request
LL_DMAMUX_REQ_USART1_TX	DMAMUX USART1 TX request
LL_DMAMUX_REQ_USART2_RX	DMAMUX USART2 RX request
LL_DMAMUX_REQ_USART2_TX	DMAMUX USART2 TX request
LL_DMAMUX_REQ_USART3_RX	DMAMUX USART3 RX request
LL_DMAMUX_REQ_USART3_TX	DMAMUX USART3 TX request
LL_DMAMUX_REQ_UART4_RX	DMAMUX UART4 RX request
LL_DMAMUX_REQ_UART4_TX	DMAMUX UART4 TX request
LL_DMAMUX_REQ_UART5_RX	DMAMUX UART5 RX request
LL_DMAMUX_REQ_UART5_TX	DMAMUX UART5 TX request
LL_DMAMUX_REQ_LPUART1_RX	DMAMUX LPUART1 RX request
LL_DMAMUX_REQ_LPUART1_TX	DMAMUX LPUART1 TX request
LL_DMAMUX_REQ_SAI1_A	DMAMUX SAI1 A request
LL_DMAMUX_REQ_SAI1_B	DMAMUX SAI1 B request
LL_DMAMUX_REQ_SAI2_A	DMAMUX SAI2 A request
LL_DMAMUX_REQ_SAI2_B	DMAMUX SAI2 B request
LL_DMAMUX_REQ_OSPI1	DMAMUX OCTOSPI1 request
LL_DMAMUX_REQ_OSPI2	DMAMUX OCTOSPI2 request
LL_DMAMUX_REQ_TIM1_CH1	DMAMUX TIM1 CH1 request
LL_DMAMUX_REQ_TIM1_CH2	DMAMUX TIM1 CH2 request
LL_DMAMUX_REQ_TIM1_CH3	DMAMUX TIM1 CH3 request
LL_DMAMUX_REQ_TIM1_CH4	DMAMUX TIM1 CH4 request
LL_DMAMUX_REQ_TIM1_UP	DMAMUX TIM1 UP request
LL_DMAMUX_REQ_TIM1_TRIG	DMAMUX TIM1 TRIG request
LL_DMAMUX_REQ_TIM1_COM	DMAMUX TIM1 COM request
LL_DMAMUX_REQ_TIM8_CH1	DMAMUX TIM8 CH1 request
LL_DMAMUX_REQ_TIM8_CH2	DMAMUX TIM8 CH2 request

LL_DMAMUX_REQ_TIM8_CH3	DMAMUX TIM8 CH3 request
LL_DMAMUX_REQ_TIM8_CH4	DMAMUX TIM8 CH4 request
LL_DMAMUX_REQ_TIM8_UP	DMAMUX TIM8 UP request
LL_DMAMUX_REQ_TIM8_TRIG	DMAMUX TIM8 TRIG request
LL_DMAMUX_REQ_TIM8_COM	DMAMUX TIM8 COM request
LL_DMAMUX_REQ_TIM2_CH1	DMAMUX TIM2 CH1 request
LL_DMAMUX_REQ_TIM2_CH2	DMAMUX TIM2 CH2 request
LL_DMAMUX_REQ_TIM2_CH3	DMAMUX TIM2 CH3 request
LL_DMAMUX_REQ_TIM2_CH4	DMAMUX TIM2 CH4 request
LL_DMAMUX_REQ_TIM2_UP	DMAMUX TIM2 UP request
LL_DMAMUX_REQ_TIM3_CH1	DMAMUX TIM3 CH1 request
LL_DMAMUX_REQ_TIM3_CH2	DMAMUX TIM3 CH2 request
LL_DMAMUX_REQ_TIM3_CH3	DMAMUX TIM3 CH3 request
LL_DMAMUX_REQ_TIM3_CH4	DMAMUX TIM3 CH4 request
LL_DMAMUX_REQ_TIM3_UP	DMAMUX TIM3 UP request
LL_DMAMUX_REQ_TIM3_TRIG	DMAMUX TIM3 TRIG request
LL_DMAMUX_REQ_TIM4_CH1	DMAMUX TIM4 CH1 request
LL_DMAMUX_REQ_TIM4_CH2	DMAMUX TIM4 CH2 request
LL_DMAMUX_REQ_TIM4_CH3	DMAMUX TIM4 CH3 request
LL_DMAMUX_REQ_TIM4_CH4	DMAMUX TIM4 CH4 request
LL_DMAMUX_REQ_TIM4_UP	DMAMUX TIM4 UP request
LL_DMAMUX_REQ_TIM5_CH1	DMAMUX TIM5 CH1 request
LL_DMAMUX_REQ_TIM5_CH2	DMAMUX TIM5 CH2 request
LL_DMAMUX_REQ_TIM5_CH3	DMAMUX TIM5 CH3 request
LL_DMAMUX_REQ_TIM5_CH4	DMAMUX TIM5 CH4 request
LL_DMAMUX_REQ_TIM5_UP	DMAMUX TIM5 UP request
LL_DMAMUX_REQ_TIM5_TRIG	DMAMUX TIM5 TRIG request
LL_DMAMUX_REQ_TIM15_CH1	DMAMUX TIM15 CH1 request
LL_DMAMUX_REQ_TIM15_UP	DMAMUX TIM15 UP request
LL_DMAMUX_REQ_TIM15_TRIG	DMAMUX TIM15 TRIG request
LL_DMAMUX_REQ_TIM15_COM	DMAMUX TIM15 COM request
LL_DMAMUX_REQ_TIM16_CH1	DMAMUX TIM16 CH1 request
LL_DMAMUX_REQ_TIM16_UP	DMAMUX TIM16 UP request
LL_DMAMUX_REQ_TIM17_CH1	DMAMUX TIM17 CH1 request
LL_DMAMUX_REQ_TIM17_UP	DMAMUX TIM17 UP request
LL_DMAMUX_REQ_DFSDM1_FLT0	DMAMUX DFSDM1_FLT0 request

LL_DMAMUX_REQ_DFSDM1_FLT1	DMAMUX DFSDM1_FLT1 request
LL_DMAMUX_REQ_DFSDM1_FLT2	DMAMUX DFSDM1_FLT2 request
LL_DMAMUX_REQ_DFSDM1_FLT3	DMAMUX DFSDM1_FLT3 request
LL_DMAMUX_REQ_DCMI	DMAMUX DCMI request
LL_DMAMUX_REQ_AES_IN	DMAMUX AES_IN request
LL_DMAMUX_REQ_AES_OUT	DMAMUX AES_OUT request
LL_DMAMUX_REQ_HASH_IN	DMAMUX HASH_IN request

External Request Signal Generation

LL_DMAMUX_REQ_GEN_EXTI_LINE0	Request signal generation from EXTI Line0
LL_DMAMUX_REQ_GEN_EXTI_LINE1	Request signal generation from EXTI Line1
LL_DMAMUX_REQ_GEN_EXTI_LINE2	Request signal generation from EXTI Line2
LL_DMAMUX_REQ_GEN_EXTI_LINE3	Request signal generation from EXTI Line3
LL_DMAMUX_REQ_GEN_EXTI_LINE4	Request signal generation from EXTI Line4
LL_DMAMUX_REQ_GEN_EXTI_LINE5	Request signal generation from EXTI Line5
LL_DMAMUX_REQ_GEN_EXTI_LINE6	Request signal generation from EXTI Line6
LL_DMAMUX_REQ_GEN_EXTI_LINE7	Request signal generation from EXTI Line7
LL_DMAMUX_REQ_GEN_EXTI_LINE8	Request signal generation from EXTI Line8
LL_DMAMUX_REQ_GEN_EXTI_LINE9	Request signal generation from EXTI Line9
LL_DMAMUX_REQ_GEN_EXTI_LINE10	Request signal generation from EXTI Line10
LL_DMAMUX_REQ_GEN_EXTI_LINE11	Request signal generation from EXTI Line11
LL_DMAMUX_REQ_GEN_EXTI_LINE12	Request signal generation from EXTI Line12
LL_DMAMUX_REQ_GEN_EXTI_LINE13	Request signal generation from EXTI Line13
LL_DMAMUX_REQ_GEN_EXTI_LINE14	Request signal generation from EXTI Line14
LL_DMAMUX_REQ_GEN_EXTI_LINE15	Request signal generation from EXTI Line15
LL_DMAMUX_REQ_GEN_DMAMUX_CH0	Request signal generation from DMAMUX channel0 Event

LL_DMAMUX_REQ_GEN_DMAMUX_CH1	Request signal generation from DMAMUX channel1 Event
LL_DMAMUX_REQ_GEN_DMAMUX_CH2	Request signal generation from DMAMUX channel2 Event
LL_DMAMUX_REQ_GEN_DMAMUX_CH3	Request signal generation from DMAMUX channel3 Event
LL_DMAMUX_REQ_GEN_LPTIM1_OUT	Request signal generation from LPTIM1 Ouput
LL_DMAMUX_REQ_GEN_LPTIM2_OUT	Request signal generation from LPTIM2 Ouput
LL_DMAMUX_REQ_GEN_DSI_TE	Request signal generation from DSI Tearing Effect
LL_DMAMUX_REQ_GEN_DSI_REFRESH_END	Request signal generation from DSI End of Refresh
LL_DMAMUX_REQ_GEN_DMA2D_TX_END	Request signal generation from DMA2D End of Transfer
LL_DMAMUX_REQ_GEN_LTDC_LINE_IT	Request signal generation from LTDC Line Interrupt

Request Generator Channel

LL_DMAMUX_REQ_GEN_0

LL_DMAMUX_REQ_GEN_1

LL_DMAMUX_REQ_GEN_2

LL_DMAMUX_REQ_GEN_3

External Request Signal Generation Polarity

LL_DMAMUX_REQ_GEN_NO_EVENT	No external DMA request generation
LL_DMAMUX_REQ_GEN_POL_RISING	External DMA request generation on event on rising edge
LL_DMAMUX_REQ_GEN_POL_FALLING	External DMA request generation on event on falling edge
LL_DMAMUX_REQ_GEN_POL_RISING_FALLING	External DMA request generation on rising and falling edge

Synchronization Signal Event

LL_DMAMUX_SYNC_EXTI_LINE0	Synchronization signal from EXTI Line0
LL_DMAMUX_SYNC_EXTI_LINE1	Synchronization signal from EXTI Line1
LL_DMAMUX_SYNC_EXTI_LINE2	Synchronization signal from EXTI Line2
LL_DMAMUX_SYNC_EXTI_LINE3	Synchronization signal from EXTI Line3
LL_DMAMUX_SYNC_EXTI_LINE4	Synchronization signal from EXTI Line4
LL_DMAMUX_SYNC_EXTI_LINE5	Synchronization signal from EXTI Line5
LL_DMAMUX_SYNC_EXTI_LINE6	Synchronization signal from EXTI Line6
LL_DMAMUX_SYNC_EXTI_LINE7	Synchronization signal from EXTI Line7

LL_DMAMUX_SYNC_EXTI_LINE8	Synchronization signal from EXTI Line8
LL_DMAMUX_SYNC_EXTI_LINE9	Synchronization signal from EXTI Line9
LL_DMAMUX_SYNC_EXTI_LINE10	Synchronization signal from EXTI Line10
LL_DMAMUX_SYNC_EXTI_LINE11	Synchronization signal from EXTI Line11
LL_DMAMUX_SYNC_EXTI_LINE12	Synchronization signal from EXTI Line12
LL_DMAMUX_SYNC_EXTI_LINE13	Synchronization signal from EXTI Line13
LL_DMAMUX_SYNC_EXTI_LINE14	Synchronization signal from EXTI Line14
LL_DMAMUX_SYNC_EXTI_LINE15	Synchronization signal from EXTI Line15
LL_DMAMUX_SYNC_DMAMUX_CH0	Synchronization signal from DMAMUX channel0 Event
LL_DMAMUX_SYNC_DMAMUX_CH1	Synchronization signal from DMAMUX channel1 Event
LL_DMAMUX_SYNC_DMAMUX_CH2	Synchronization signal from DMAMUX channel2 Event
LL_DMAMUX_SYNC_DMAMUX_CH3	Synchronization signal from DMAMUX channel3 Event
LL_DMAMUX_SYNC_LPTIM1_OUT	Synchronization signal from LPTIM1 Ouput
LL_DMAMUX_SYNC_LPTIM2_OUT	Synchronization signal from LPTIM2 Ouput
LL_DMAMUX_SYNC_DSI_TE	Synchronization signal from DSI Tearing Effect
LL_DMAMUX_SYNC_DSI_REFRESH_END	Synchronization signal from DSI End of Refresh
LL_DMAMUX_SYNC_DMA2D_TX_END	Synchronization signal from DMA2D End of Transfer
LL_DMAMUX_SYNC_LTDC_LINE_IT	Synchronization signal from LTDC Line Interrupt

Synchronization Signal Polarity

LL_DMAMUX_SYNC_NO_EVENT	All requests are blocked
LL_DMAMUX_SYNC_POL_RISING	Synchronization on event on rising edge
LL_DMAMUX_SYNC_POL_FALLING	Synchronization on event on falling edge
LL_DMAMUX_SYNC_POL_RISING_FALLING	Synchronization on event on rising and falling edge

Common Write and read registers macros

LL_DMAMUX_WriteReg

Description:

- Write a value in DMAMUX register.

Parameters:

- __INSTANCE__: DMAMUX Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMAMUX_ReadReg

Description:

- Read a value in DMAMUX register.

Parameters:

- __INSTANCE__: DMAMUX Instance
- __REG__: Register to be read

Return value:

- Register: value

82 LL DMA Generic Driver

82.1 DMA Firmware driver registers structures

82.1.1 LL_DMA_InitTypeDef

Data Fields

- ***uint32_t PeriphOrM2MSrcAddress***
- ***uint32_t MemoryOrM2MDstAddress***
- ***uint32_t Direction***
- ***uint32_t Mode***
- ***uint32_t PeriphOrM2MSrcIncMode***
- ***uint32_t MemoryOrM2MDstIncMode***
- ***uint32_t PeriphOrM2MSrcDataSize***
- ***uint32_t MemoryOrM2MDstDataSize***
- ***uint32_t NbData***
- ***uint32_t PeriphRequest***
- ***uint32_t Priority***

Field Documentation

- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcAddress***
  Specifies the peripheral base address for DMA transfer or as Source base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstAddress***
  Specifies the memory base address for DMA transfer or as Destination base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitTypeDef::Direction***
  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_LL_EC_DIRECTION***This feature can be modified afterwards using unitary function **LL_DMA_SetDataTransferDirection()**.
- ***uint32_t LL_DMA_InitTypeDef::Mode***
  Specifies the normal or circular operation mode. This parameter can be a value of ***DMA_LL_EC_MODE***
  Note:: The circular buffer mode cannot be used if the memory to memory data transfer direction is configured on the selected Channel This feature can be modified afterwards using unitary function **LL_DMA_SetMode()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcIncMode***
  Specifies whether the Peripheral address or Source address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***DMA_LL_EC_PERIPH***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstIncMode***
  Specifies whether the Memory address or Destination address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***DMA_LL_EC_MEMORY***This feature can be modified afterwards using unitary function **LL_DMA_SetMemoryIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcDataSize***
  Specifies the Peripheral data size alignment or Source data size alignment (byte, half

word, word) in case of memory to memory transfer direction. This parameter can be a value of ***DMA_LL_EC_PDATAALIGN***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphSize()**.

- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstDataSize***
  Specifies the Memory data size alignment or Destination data size alignment (byte, half word, word) in case of memory to memory transfer direction. This parameter can be a value of ***DMA_LL_EC_MDATAALIGN***This feature can be modified afterwards using unitary function **LL_DMA_SetMemorySize()**.
- ***uint32_t LL_DMA_InitTypeDef::NbData***
  Specifies the number of data to transfer, in data unit. The data unit is equal to the source buffer configuration set in PeripheralSize or MemorySize parameters depending in the transfer direction. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFFThis feature can be modified afterwards using unitary function **LL_DMA_SetDataLength()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphRequest***
  Specifies the peripheral request. This parameter can be a value of ***DMAMUX_LL_EC_REQUEST***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphRequest()**.
- ***uint32_t LL_DMA_InitTypeDef::Priority***
  Specifies the channel priority level. This parameter can be a value of ***DMA_LL_EC_PRIORITY***This feature can be modified afterwards using unitary function **LL_DMA_SetChannelPriorityLevel()**.

82.2 DMA Firmware driver API description

82.2.1 Detailed description of functions

LL_DMA_EnableChannel

Function name	**__STATIC_INLINE void LL_DMA_EnableChannel (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Enable DMA channel.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR EN LL_DMA_EnableChannel

LL_DMA_DisableChannel

Function name	**__STATIC_INLINE void LL_DMA_DisableChannel (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Disable DMA channel.

Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR EN LL_DMA_DisableChannel

LL_DMA_IsEnabledChannel

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannel (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Check if DMA channel is enabled or disabled.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCR EN LL_DMA_IsEnabledChannel

LL_DMA_ConfigTransfer

Function name	**__STATIC_INLINE void LL_DMA_ConfigTransfer (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)**
Function description	Configure all parameters link to DMA transfer.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **Configuration:** This parameter must be a combination of all

	the following values: – LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH or LL_DMA_DIRECTION_MEMORY_TO_MEMORY – LL_DMA_MODE_NORMAL or LL_DMA_MODE_CIRCULAR – LL_DMA_PERIPH_INCREMENT or LL_DMA_PERIPH_NOINCREMENT – LL_DMA_MEMORY_INCREMENT or LL_DMA_MEMORY_NOINCREMENT – LL_DMA_PDATAALIGN_BYTE or LL_DMA_PDATAALIGN_HALFWORD or LL_DMA_PDATAALIGN_WORD – LL_DMA_MDATAALIGN_BYTE or LL_DMA_MDATAALIGN_HALFWORD or LL_DMA_MDATAALIGN_WORD – LL_DMA_PRIORITY_LOW or LL_DMA_PRIORITY_MEDIUM or LL_DMA_PRIORITY_HIGH or LL_DMA_PRIORITY_VERYHIGH
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR DIR LL_DMA_ConfigTransfer • CCR MEM2MEM LL_DMA_ConfigTransfer • CCR CIRC LL_DMA_ConfigTransfer • CCR PINC LL_DMA_ConfigTransfer • CCR MINC LL_DMA_ConfigTransfer • CCR PSIZE LL_DMA_ConfigTransfer • CCR MSIZE LL_DMA_ConfigTransfer • CCR PL LL_DMA_ConfigTransfer

LL_DMA_SetDataTransferDirection

Function name	**__STATIC_INLINE void LL_DMA_SetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Direction)**
Function description	Set Data transfer direction (read from peripheral or from memory).
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **Direction:** This parameter can be one of the following values: – LL_DMA_DIRECTION_PERIPH_TO_MEMORY – LL_DMA_DIRECTION_MEMORY_TO_PERIPH – LL_DMA_DIRECTION_MEMORY_TO_MEMORY
Return values	• **None:**

Reference Manual to LL API cross reference:	• CCR DIR LL_DMA_SetDataTransferDirection • CCR MEM2MEM LL_DMA_SetDataTransferDirection

LL_DMA_GetDataTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Data transfer direction (read from peripheral or from memory).
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMA_DIRECTION_PERIPH_TO_MEMORY – LL_DMA_DIRECTION_MEMORY_TO_PERIPH – LL_DMA_DIRECTION_MEMORY_TO_MEMORY
Reference Manual to LL API cross reference:	• CCR DIR LL_DMA_GetDataTransferDirection • CCR MEM2MEM LL_DMA_GetDataTransferDirection

LL_DMA_SetMode

Function name	**__STATIC_INLINE void LL_DMA_SetMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Mode)**
Function description	Set DMA mode circular or normal.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **Mode:** This parameter can be one of the following values: – LL_DMA_MODE_NORMAL – LL_DMA_MODE_CIRCULAR
Return values	• **None:**
Notes	• The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel.
Reference Manual to LL API cross	• CCR CIRC LL_DMA_SetMode

reference:

LL_DMA_GetMode

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetMode (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get DMA mode circular or normal.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMA_MODE_NORMAL – LL_DMA_MODE_CIRCULAR
Reference Manual to LL API cross reference:	• CCR CIRC LL_DMA_GetMode

LL_DMA_SetPeriphIncMode

Function name	**__STATIC_INLINE void LL_DMA_SetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcIncMode)**
Function description	Set Peripheral increment mode.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **PeriphOrM2MSrcIncMode:** This parameter can be one of the following values: – LL_DMA_PERIPH_INCREMENT – LL_DMA_PERIPH_NOINCREMENT
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR PINC LL_DMA_SetPeriphIncMode

LL_DMA_GetPeriphIncMode

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Peripheral increment mode.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMA_PERIPH_INCREMENT – LL_DMA_PERIPH_NOINCREMENT
Reference Manual to LL API cross reference:	• CCR PINC LL_DMA_GetPeriphIncMode

LL_DMA_SetMemoryIncMode

Function name	**__STATIC_INLINE void LL_DMA_SetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstIncMode)**
Function description	Set Memory increment mode.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **MemoryOrM2MDstIncMode:** This parameter can be one of the following values: – LL_DMA_MEMORY_INCREMENT – LL_DMA_MEMORY_NOINCREMENT
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR MINC LL_DMA_SetMemoryIncMode

LL_DMA_GetMemoryIncMode

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Channel)**

Function description	Get Memory increment mode.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMA_MEMORY_INCREMENT – LL_DMA_MEMORY_NOINCREMENT
Reference Manual to LL API cross reference:	• CCR MINC LL_DMA_GetMemoryIncMode

LL_DMA_SetPeriphSize

Function name	**__STATIC_INLINE void LL_DMA_SetPeriphSize (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcDataSize)**
Function description	Set Peripheral size.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **PeriphOrM2MSrcDataSize:** This parameter can be one of the following values: – LL_DMA_PDATAALIGN_BYTE – LL_DMA_PDATAALIGN_HALFWORD – LL_DMA_PDATAALIGN_WORD
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR PSIZE LL_DMA_SetPeriphSize

LL_DMA_GetPeriphSize

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetPeriphSize (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Peripheral size.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values:

- – LL_DMA_CHANNEL_1
- – LL_DMA_CHANNEL_2
- – LL_DMA_CHANNEL_3
- – LL_DMA_CHANNEL_4
- – LL_DMA_CHANNEL_5
- – LL_DMA_CHANNEL_6
- – LL_DMA_CHANNEL_7

Return values	• **Returned:** value can be one of the following values: – LL_DMA_PDATAALIGN_BYTE – LL_DMA_PDATAALIGN_HALFWORD – LL_DMA_PDATAALIGN_WORD
Reference Manual to LL API cross reference:	• CCR PSIZE LL_DMA_GetPeriphSize

LL_DMA_SetMemorySize

Function name	**__STATIC_INLINE void LL_DMA_SetMemorySize (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstDataSize)**
Function description	Set Memory size.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **MemoryOrM2MDstDataSize:** This parameter can be one of the following values: – LL_DMA_MDATAALIGN_BYTE – LL_DMA_MDATAALIGN_HALFWORD – LL_DMA_MDATAALIGN_WORD
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR MSIZE LL_DMA_SetMemorySize

LL_DMA_GetMemorySize

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetMemorySize (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Memory size.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2

	– LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMA_MDATAALIGN_BYTE – LL_DMA_MDATAALIGN_HALFWORD – LL_DMA_MDATAALIGN_WORD
Reference Manual to LL API cross reference:	• CCR MSIZE LL_DMA_GetMemorySize

LL_DMA_SetChannelPriorityLevel

Function name	**__STATIC_INLINE void LL_DMA_SetChannelPriorityLevel (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Priority)**
Function description	Set Channel priority level.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **Priority:** This parameter can be one of the following values: – LL_DMA_PRIORITY_LOW – LL_DMA_PRIORITY_MEDIUM – LL_DMA_PRIORITY_HIGH – LL_DMA_PRIORITY_VERYHIGH
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR PL LL_DMA_SetChannelPriorityLevel

LL_DMA_GetChannelPriorityLevel

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetChannelPriorityLevel (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Channel priority level.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5

	– LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMA_PRIORITY_LOW – LL_DMA_PRIORITY_MEDIUM – LL_DMA_PRIORITY_HIGH – LL_DMA_PRIORITY_VERYHIGH
Reference Manual to LL API cross reference:	• CCR PL LL_DMA_GetChannelPriorityLevel

LL_DMA_SetDataLength

Function name	**__STATIC_INLINE void LL_DMA_SetDataLength (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t NbData)**
Function description	Set Number of data to transfer.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **NbData:** Between Min_Data = 0 and Max_Data = 0x0000FFFF
Return values	• **None:**
Notes	• This action has no effect if channel is enabled.
Reference Manual to LL API cross reference:	• CNDTR NDT LL_DMA_SetDataLength

LL_DMA_GetDataLength

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetDataLength (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Number of data to transfer.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes	• Once the channel is enabled, the return value indicate the remaining bytes to be transmitted.
Reference Manual to LL API cross reference:	• CNDTR NDT LL_DMA_GetDataLength

LL_DMA_ConfigAddresses

Function name	**__STATIC_INLINE void LL_DMA_ConfigAddresses (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DstAddress, uint32_t Direction)**
Function description	Configure the Source and Destination addresses.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **SrcAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF • **DstAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF • **Direction:** This parameter can be one of the following values: – LL_DMA_DIRECTION_PERIPH_TO_MEMORY – LL_DMA_DIRECTION_MEMORY_TO_PERIPH – LL_DMA_DIRECTION_MEMORY_TO_MEMORY
Return values	• **None:**
Notes	• This API must not be called when the DMA channel is enabled. • Each IP using DMA provides an API to get directly the register adress (LL_PPP_DMA_GetRegAddr).
Reference Manual to LL API cross reference:	• CPAR PA LL_DMA_ConfigAddresses • CMAR MA LL_DMA_ConfigAddresses

LL_DMA_SetMemoryAddress

Function name	**__STATIC_INLINE void LL_DMA_SetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)**
Function description	Set the Memory address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3

	– LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
Return values	• **None:**
Notes	• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only. • This API must not be called when the DMA channel is enabled.
Reference Manual to LL API cross reference:	• CMAR MA LL_DMA_SetMemoryAddress

LL_DMA_SetPeriphAddress

Function name	**__STATIC_INLINE void LL_DMA_SetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphAddress)**
Function description	Set the Peripheral address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **PeriphAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
Return values	• **None:**
Notes	• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only. • This API must not be called when the DMA channel is enabled.
Reference Manual to LL API cross reference:	• CPAR PA LL_DMA_SetPeriphAddress

LL_DMA_GetMemoryAddress

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Memory address.

Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
Reference Manual to LL API cross reference:	• CMAR MA LL_DMA_GetMemoryAddress

LL_DMA_GetPeriphAddress

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get Peripheral address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
Reference Manual to LL API cross reference:	• CPAR PA LL_DMA_GetPeriphAddress

LL_DMA_SetM2MSrcAddress

Function name	**__STATIC_INLINE void LL_DMA_SetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)**
Function description	Set the Memory to Memory Source address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1

- – LL_DMA_CHANNEL_2
- – LL_DMA_CHANNEL_3
- – LL_DMA_CHANNEL_4
- – LL_DMA_CHANNEL_5
- – LL_DMA_CHANNEL_6
- – LL_DMA_CHANNEL_7
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values	• **None:**
Notes	• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only. • This API must not be called when the DMA channel is enabled.
Reference Manual to LL API cross reference:	• CPAR PA LL_DMA_SetM2MSrcAddress

LL_DMA_SetM2MDstAddress

Function name	**__STATIC_INLINE void LL_DMA_SetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)**
Function description	Set the Memory to Memory Destination address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7 • **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
Return values	• **None:**
Notes	• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only. • This API must not be called when the DMA channel is enabled.
Reference Manual to LL API cross reference:	• CMAR MA LL_DMA_SetM2MDstAddress

LL_DMA_GetM2MSrcAddress

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get the Memory to Memory Source address.

Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
Reference Manual to LL API cross reference:	• CPAR PA LL_DMA_GetM2MSrcAddress

LL_DMA_GetM2MDstAddress

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get the Memory to Memory Destination address.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF
Notes	• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
Reference Manual to LL API cross reference:	• CMAR MA LL_DMA_GetM2MDstAddress

LL_DMA_SetPeriphRequest

Function name	**__STATIC_INLINE void LL_DMA_SetPeriphRequest (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Request)**
Function description	Set DMA request for DMA Channels on DMAMUX Channel x.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4

- LL_DMA_CHANNEL_5
- LL_DMA_CHANNEL_6
- LL_DMA_CHANNEL_7

- **Request:** This parameter can be one of the following values:
  - LL_DMAMUX_REQUEST_MEM2MEM
  - LL_DMAMUX_REQUEST_GENERATOR0
  - LL_DMAMUX_REQUEST_GENERATOR1
  - LL_DMAMUX_REQUEST_GENERATOR2
  - LL_DMAMUX_REQUEST_GENERATOR3
  - LL_DMAMUX_REQUEST_ADC1
  - LL_DMAMUX_REQUEST_DAC1_CH1
  - LL_DMAMUX_REQUEST_DAC1_CH2
  - LL_DMAMUX_REQUEST_TIM6_UP
  - LL_DMAMUX_REQUEST_TIM7_UP
  - LL_DMAMUX_REQUEST_SPI1_RX
  - LL_DMAMUX_REQUEST_SPI1_TX
  - LL_DMAMUX_REQUEST_SPI2_RX
  - LL_DMAMUX_REQUEST_SPI2_TX
  - LL_DMAMUX_REQUEST_SPI3_RX
  - LL_DMAMUX_REQUEST_SPI3_TX
  - LL_DMAMUX_REQUEST_I2C1_RX
  - LL_DMAMUX_REQUEST_I2C1_TX
  - LL_DMAMUX_REQUEST_I2C2_RX
  - LL_DMAMUX_REQUEST_I2C2_TX
  - LL_DMAMUX_REQUEST_I2C3_RX
  - LL_DMAMUX_REQUEST_I2C3_TX
  - LL_DMAMUX_REQUEST_I2C4_RX
  - LL_DMAMUX_REQUEST_I2C4_TX
  - LL_DMAMUX_REQUEST_USART1_RX
  - LL_DMAMUX_REQUEST_USART1_TX
  - LL_DMAMUX_REQUEST_USART2_RX
  - LL_DMAMUX_REQUEST_USART2_TX
  - LL_DMAMUX_REQUEST_USART3_RX
  - LL_DMAMUX_REQUEST_USART3_TX
  - LL_DMAMUX_REQUEST_UART4_RX
  - LL_DMAMUX_REQUEST_UART4_TX
  - LL_DMAMUX_REQUEST_UART5_RX
  - LL_DMAMUX_REQUEST_UART5_TX
  - LL_DMAMUX_REQUEST_LPUART1_RX
  - LL_DMAMUX_REQUEST_LPUART1_TX
  - LL_DMAMUX_REQUEST_SAI1_A
  - LL_DMAMUX_REQUEST_SAI1_B
  - LL_DMAMUX_REQUEST_SAI2_A
  - LL_DMAMUX_REQUEST_SAI2_B
  - LL_DMAMUX_REQUEST_OSPI1
  - LL_DMAMUX_REQUEST_OSPI2
  - LL_DMAMUX_REQUEST_TIM1_CH1
  - LL_DMAMUX_REQUEST_TIM1_CH2
  - LL_DMAMUX_REQUEST_TIM1_CH3
  - LL_DMAMUX_REQUEST_TIM1_CH4
  - LL_DMAMUX_REQUEST_TIM1_UP
  - LL_DMAMUX_REQUEST_TIM1_TRIG

- LL_DMAMUX_REQUEST_TIM1_COM
- LL_DMAMUX_REQUEST_TIM8_CH1
- LL_DMAMUX_REQUEST_TIM8_CH2
- LL_DMAMUX_REQUEST_TIM8_CH3
- LL_DMAMUX_REQUEST_TIM8_CH4
- LL_DMAMUX_REQUEST_TIM8_UP
- LL_DMAMUX_REQUEST_TIM8_TRIG
- LL_DMAMUX_REQUEST_TIM8_COM
- LL_DMAMUX_REQUEST_TIM2_CH1
- LL_DMAMUX_REQUEST_TIM2_CH2
- LL_DMAMUX_REQUEST_TIM2_CH3
- LL_DMAMUX_REQUEST_TIM2_CH4
- LL_DMAMUX_REQUEST_TIM2_UP
- LL_DMAMUX_REQUEST_TIM3_CH1
- LL_DMAMUX_REQUEST_TIM3_CH2
- LL_DMAMUX_REQUEST_TIM3_CH3
- LL_DMAMUX_REQUEST_TIM3_CH4
- LL_DMAMUX_REQUEST_TIM3_UP
- LL_DMAMUX_REQUEST_TIM3_TRIG
- LL_DMAMUX_REQUEST_TIM4_CH1
- LL_DMAMUX_REQUEST_TIM4_CH2
- LL_DMAMUX_REQUEST_TIM4_CH3
- LL_DMAMUX_REQUEST_TIM4_CH4
- LL_DMAMUX_REQUEST_TIM4_UP
- LL_DMAMUX_REQUEST_TIM5_CH1
- LL_DMAMUX_REQUEST_TIM5_CH2
- LL_DMAMUX_REQUEST_TIM5_CH3
- LL_DMAMUX_REQUEST_TIM5_CH4
- LL_DMAMUX_REQUEST_TIM5_UP
- LL_DMAMUX_REQUEST_TIM5_TRIG
- LL_DMAMUX_REQUEST_TIM15_CH1
- LL_DMAMUX_REQUEST_TIM15_UP
- LL_DMAMUX_REQUEST_TIM15_TRIG
- LL_DMAMUX_REQUEST_TIM15_COM
- LL_DMAMUX_REQUEST_TIM16_CH1
- LL_DMAMUX_REQUEST_TIM16_UP
- LL_DMAMUX_REQUEST_TIM17_CH1
- LL_DMAMUX_REQUEST_TIM17_UP
- LL_DMAMUX_REQUEST_DFSDM1_FLT0
- LL_DMAMUX_REQUEST_DFSDM1_FLT1
- LL_DMAMUX_REQUEST_DFSDM1_FLT2
- LL_DMAMUX_REQUEST_DFSDM1_FLT3
- LL_DMAMUX_REQUEST_DCMI
- LL_DMAMUX_REQUEST_AES_IN
- LL_DMAMUX_REQUEST_AES_OUT
- LL_DMAMUX_REQUEST_HASH_IN

Return values
- **None:**

Notes
- DMAMUX channel 0 to 6 are mapped to DMA1 channel 1 to 7. DMAMUX channel 7 to 13 are mapped to DMA2 channel 1 to 7.

Reference Manual to LL API cross reference:	• CxCR DMAREQ_ID LL_DMA_SetPeriphRequest

LL_DMA_GetPeriphRequest

Function name	**__STATIC_INLINE uint32_t LL_DMA_GetPeriphRequest (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Get DMA request for DMA Channels on DMAMUX Channel x.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **Returned:** value can be one of the following values: – LL_DMAMUX_REQUEST_MEM2MEM – LL_DMAMUX_REQUEST_GENERATOR0 – LL_DMAMUX_REQUEST_GENERATOR1 – LL_DMAMUX_REQUEST_GENERATOR2 – LL_DMAMUX_REQUEST_GENERATOR3 – LL_DMAMUX_REQUEST_ADC1 – LL_DMAMUX_REQUEST_DAC1_CH1 – LL_DMAMUX_REQUEST_DAC1_CH2 – LL_DMAMUX_REQUEST_TIM6_UP – LL_DMAMUX_REQUEST_TIM7_UP – LL_DMAMUX_REQUEST_SPI1_RX – LL_DMAMUX_REQUEST_SPI1_TX – LL_DMAMUX_REQUEST_SPI2_RX – LL_DMAMUX_REQUEST_SPI2_TX – LL_DMAMUX_REQUEST_SPI3_RX – LL_DMAMUX_REQUEST_SPI3_TX – LL_DMAMUX_REQUEST_I2C1_RX – LL_DMAMUX_REQUEST_I2C1_TX – LL_DMAMUX_REQUEST_I2C2_RX – LL_DMAMUX_REQUEST_I2C2_TX – LL_DMAMUX_REQUEST_I2C3_RX – LL_DMAMUX_REQUEST_I2C3_TX – LL_DMAMUX_REQUEST_I2C4_RX – LL_DMAMUX_REQUEST_I2C4_TX – LL_DMAMUX_REQUEST_USART1_RX – LL_DMAMUX_REQUEST_USART1_TX – LL_DMAMUX_REQUEST_USART2_RX – LL_DMAMUX_REQUEST_USART2_TX – LL_DMAMUX_REQUEST_USART3_RX – LL_DMAMUX_REQUEST_USART3_TX – LL_DMAMUX_REQUEST_UART4_RX – LL_DMAMUX_REQUEST_UART4_TX

- LL_DMAMUX_REQUEST_UART5_RX
- LL_DMAMUX_REQUEST_UART5_TX
- LL_DMAMUX_REQUEST_LPUART1_RX
- LL_DMAMUX_REQUEST_LPUART1_TX
- LL_DMAMUX_REQUEST_SAI1_A
- LL_DMAMUX_REQUEST_SAI1_B
- LL_DMAMUX_REQUEST_SAI2_A
- LL_DMAMUX_REQUEST_SAI2_B
- LL_DMAMUX_REQUEST_OSPI1
- LL_DMAMUX_REQUEST_OSPI2
- LL_DMAMUX_REQUEST_TIM1_CH1
- LL_DMAMUX_REQUEST_TIM1_CH2
- LL_DMAMUX_REQUEST_TIM1_CH3
- LL_DMAMUX_REQUEST_TIM1_CH4
- LL_DMAMUX_REQUEST_TIM1_UP
- LL_DMAMUX_REQUEST_TIM1_TRIG
- LL_DMAMUX_REQUEST_TIM1_COM
- LL_DMAMUX_REQUEST_TIM8_CH1
- LL_DMAMUX_REQUEST_TIM8_CH2
- LL_DMAMUX_REQUEST_TIM8_CH3
- LL_DMAMUX_REQUEST_TIM8_CH4
- LL_DMAMUX_REQUEST_TIM8_UP
- LL_DMAMUX_REQUEST_TIM8_TRIG
- LL_DMAMUX_REQUEST_TIM8_COM
- LL_DMAMUX_REQUEST_TIM2_CH1
- LL_DMAMUX_REQUEST_TIM2_CH2
- LL_DMAMUX_REQUEST_TIM2_CH3
- LL_DMAMUX_REQUEST_TIM2_CH4
- LL_DMAMUX_REQUEST_TIM2_UP
- LL_DMAMUX_REQUEST_TIM3_CH1
- LL_DMAMUX_REQUEST_TIM3_CH2
- LL_DMAMUX_REQUEST_TIM3_CH3
- LL_DMAMUX_REQUEST_TIM3_CH4
- LL_DMAMUX_REQUEST_TIM3_UP
- LL_DMAMUX_REQUEST_TIM3_TRIG
- LL_DMAMUX_REQUEST_TIM4_CH1
- LL_DMAMUX_REQUEST_TIM4_CH2
- LL_DMAMUX_REQUEST_TIM4_CH3
- LL_DMAMUX_REQUEST_TIM4_CH4
- LL_DMAMUX_REQUEST_TIM4_UP
- LL_DMAMUX_REQUEST_TIM5_CH1
- LL_DMAMUX_REQUEST_TIM5_CH2
- LL_DMAMUX_REQUEST_TIM5_CH3
- LL_DMAMUX_REQUEST_TIM5_CH4
- LL_DMAMUX_REQUEST_TIM5_UP
- LL_DMAMUX_REQUEST_TIM5_TRIG
- LL_DMAMUX_REQUEST_TIM15_CH1
- LL_DMAMUX_REQUEST_TIM15_UP
- LL_DMAMUX_REQUEST_TIM15_TRIG
- LL_DMAMUX_REQUEST_TIM15_COM
- LL_DMAMUX_REQUEST_TIM16_CH1
- LL_DMAMUX_REQUEST_TIM16_UP

- – LL_DMAMUX_REQUEST_TIM17_CH1
- – LL_DMAMUX_REQUEST_TIM17_UP
- – LL_DMAMUX_REQUEST_DFSDM1_FLT0
- – LL_DMAMUX_REQUEST_DFSDM1_FLT1
- – LL_DMAMUX_REQUEST_DFSDM1_FLT2
- – LL_DMAMUX_REQUEST_DFSDM1_FLT3
- – LL_DMAMUX_REQUEST_DCMI
- – LL_DMAMUX_REQUEST_AES_IN
- – LL_DMAMUX_REQUEST_AES_OUT
- – LL_DMAMUX_REQUEST_HASH_IN

Notes	• DMAMUX channel 0 to 6 are mapped to DMA1 channel 1 to 7. DMAMUX channel 7 to 13 are mapped to DMA2 channel 1 to 7.
Reference Manual to LL API cross reference:	• CxCR DMAREQ_ID LL_DMA_GetPeriphRequest

LL_DMA_IsActiveFlag_GI1

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF1 LL_DMA_IsActiveFlag_GI1

LL_DMA_IsActiveFlag_GI2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF2 LL_DMA_IsActiveFlag_GI2

LL_DMA_IsActiveFlag_GI3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• ISR GIF3 LL_DMA_IsActiveFlag_GI3

LL_DMA_IsActiveFlag_GI4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF4 LL_DMA_IsActiveFlag_GI4

LL_DMA_IsActiveFlag_GI5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF5 LL_DMA_IsActiveFlag_GI5

LL_DMA_IsActiveFlag_GI6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF6 LL_DMA_IsActiveFlag_GI6

LL_DMA_IsActiveFlag_GI7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• ISR GIF7 LL_DMA_IsActiveFlag_GI7

reference:

LL_DMA_IsActiveFlag_TC1

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF1 LL_DMA_IsActiveFlag_TC1

LL_DMA_IsActiveFlag_TC2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF2 LL_DMA_IsActiveFlag_TC2

LL_DMA_IsActiveFlag_TC3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF3 LL_DMA_IsActiveFlag_TC3

LL_DMA_IsActiveFlag_TC4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF4 LL_DMA_IsActiveFlag_TC4

LL_DMA_IsActiveFlag_TC5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF5 LL_DMA_IsActiveFlag_TC5

LL_DMA_IsActiveFlag_TC6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF6 LL_DMA_IsActiveFlag_TC6

LL_DMA_IsActiveFlag_TC7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF7 LL_DMA_IsActiveFlag_TC7

LL_DMA_IsActiveFlag_HT1

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF1 LL_DMA_IsActiveFlag_HT1

LL_DMA_IsActiveFlag_HT2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF2 LL_DMA_IsActiveFlag_HT2

LL_DMA_IsActiveFlag_HT3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF3 LL_DMA_IsActiveFlag_HT3

LL_DMA_IsActiveFlag_HT4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF4 LL_DMA_IsActiveFlag_HT4

LL_DMA_IsActiveFlag_HT5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF5 LL_DMA_IsActiveFlag_HT5

LL_DMA_IsActiveFlag_HT6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF6 LL_DMA_IsActiveFlag_HT6

LL_DMA_IsActiveFlag_HT7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF7 LL_DMA_IsActiveFlag_HT7

LL_DMA_IsActiveFlag_TE1

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF1 LL_DMA_IsActiveFlag_TE1

LL_DMA_IsActiveFlag_TE2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF2 LL_DMA_IsActiveFlag_TE2

LL_DMA_IsActiveFlag_TE3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF3 LL_DMA_IsActiveFlag_TE3

LL_DMA_IsActiveFlag_TE4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF4 LL_DMA_IsActiveFlag_TE4

LL_DMA_IsActiveFlag_TE5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF5 LL_DMA_IsActiveFlag_TE5

LL_DMA_IsActiveFlag_TE6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF6 LL_DMA_IsActiveFlag_TE6

LL_DMA_IsActiveFlag_TE7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF7 LL_DMA_IsActiveFlag_TE7

LL_DMA_ClearFlag_GI1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF1 LL_DMA_ClearFlag_GI1

LL_DMA_ClearFlag_GI2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF2 LL_DMA_ClearFlag_GI2

LL_DMA_ClearFlag_GI3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF3 LL_DMA_ClearFlag_GI3

LL_DMA_ClearFlag_GI4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF4 LL_DMA_ClearFlag_GI4

LL_DMA_ClearFlag_GI5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF5 LL_DMA_ClearFlag_GI5

LL_DMA_ClearFlag_GI6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF6 LL_DMA_ClearFlag_GI6

LL_DMA_ClearFlag_GI7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF7 LL_DMA_ClearFlag_GI7

LL_DMA_ClearFlag_TC1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF1 LL_DMA_ClearFlag_TC1

LL_DMA_ClearFlag_TC2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF2 LL_DMA_ClearFlag_TC2

LL_DMA_ClearFlag_TC3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF3 LL_DMA_ClearFlag_TC3

LL_DMA_ClearFlag_TC4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF4 LL_DMA_ClearFlag_TC4

LL_DMA_ClearFlag_TC5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF5 LL_DMA_ClearFlag_TC5

LL_DMA_ClearFlag_TC6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF6 LL_DMA_ClearFlag_TC6

LL_DMA_ClearFlag_TC7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF7 LL_DMA_ClearFlag_TC7

LL_DMA_ClearFlag_HT1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF1 LL_DMA_ClearFlag_HT1

LL_DMA_ClearFlag_HT2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF2 LL_DMA_ClearFlag_HT2

LL_DMA_ClearFlag_HT3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF3 LL_DMA_ClearFlag_HT3

LL_DMA_ClearFlag_HT4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF4 LL_DMA_ClearFlag_HT4

LL_DMA_ClearFlag_HT5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF5 LL_DMA_ClearFlag_HT5

LL_DMA_ClearFlag_HT6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF6 LL_DMA_ClearFlag_HT6

LL_DMA_ClearFlag_HT7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF7 LL_DMA_ClearFlag_HT7

LL_DMA_ClearFlag_TE1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF1 LL_DMA_ClearFlag_TE1

LL_DMA_ClearFlag_TE2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF2 LL_DMA_ClearFlag_TE2

LL_DMA_ClearFlag_TE3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF3 LL_DMA_ClearFlag_TE3

LL_DMA_ClearFlag_TE4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF4 LL_DMA_ClearFlag_TE4

LL_DMA_ClearFlag_TE5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF5 LL_DMA_ClearFlag_TE5

LL_DMA_ClearFlag_TE6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF6 LL_DMA_ClearFlag_TE6

LL_DMA_ClearFlag_TE7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF7 LL_DMA_ClearFlag_TE7

LL_DMA_EnableIT_TC

Function name	**__STATIC_INLINE void LL_DMA_EnableIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Enable Transfer complete interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR TCIE LL_DMA_EnableIT_TC

LL_DMA_EnableIT_HT

Function name	**__STATIC_INLINE void LL_DMA_EnableIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Enable Half transfer interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None:**
Reference Manual to LL API cross	• CCR HTIE LL_DMA_EnableIT_HT

reference:

LL_DMA_EnableIT_TE

Function name	**__STATIC_INLINE void LL_DMA_EnableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Enable Transfer error interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR TEIE LL_DMA_EnableIT_TE

LL_DMA_DisableIT_TC

Function name	**__STATIC_INLINE void LL_DMA_DisableIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Disable Transfer complete interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR TCIE LL_DMA_DisableIT_TC

LL_DMA_DisableIT_HT

Function name	**__STATIC_INLINE void LL_DMA_DisableIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Disable Half transfer interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1

- – LL_DMA_CHANNEL_2
- – LL_DMA_CHANNEL_3
- – LL_DMA_CHANNEL_4
- – LL_DMA_CHANNEL_5
- – LL_DMA_CHANNEL_6
- – LL_DMA_CHANNEL_7

Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR HTIE LL_DMA_DisableIT_HT

LL_DMA_DisableIT_TE

Function name	**__STATIC_INLINE void LL_DMA_DisableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Disable Transfer error interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR TEIE LL_DMA_DisableIT_TE

LL_DMA_IsEnabledIT_TC

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Check if Transfer complete Interrupt is enabled.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCR TCIE LL_DMA_IsEnabledIT_TC

LL_DMA_IsEnabledIT_HT

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Check if Half transfer Interrupt is enabled.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCR HTIE LL_DMA_IsEnabledIT_HT

LL_DMA_IsEnabledIT_TE

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Check if Transfer error Interrupt is enabled.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCR TEIE LL_DMA_IsEnabledIT_TE

LL_DMA_Init

Function name	**uint32_t LL_DMA_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitTypeDef * DMA_InitStruct)**
Function description	Initialize the DMA registers according to the specified parameters in DMA_InitStruct.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3

- – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
- **DMA_InitStruct:** pointer to a LL_DMA_InitTypeDef structure.

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: DMA registers are initialized
  – ERROR: Not applicable

Notes
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros: __LL_DMA_GET_INSTANCE __LL_DMA_GET_CHANNEL

LL_DMA_DeInit

Function name
uint32_t LL_DMA_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
De-initialize the DMA registers to their default reset values.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_ALL

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: DMA registers are de-initialized
  – ERROR: DMA registers are not de-initialized

LL_DMA_StructInit

Function name
void LL_DMA_StructInit (LL_DMA_InitTypeDef * DMA_InitStruct)

Function description
Set each LL_DMA_InitTypeDef field to default value.

Parameters
- **DMA_InitStruct:** Pointer to a LL_DMA_InitTypeDef structure.

Return values
- **None:**

82.3 DMA Firmware driver defines

82.3.1 DMA

CHANNEL

LL_DMA_CHANNEL_1	DMA Channel 1
LL_DMA_CHANNEL_2	DMA Channel 2

LL_DMA_CHANNEL_3	DMA Channel 3
LL_DMA_CHANNEL_4	DMA Channel 4
LL_DMA_CHANNEL_5	DMA Channel 5
LL_DMA_CHANNEL_6	DMA Channel 6
LL_DMA_CHANNEL_7	DMA Channel 7
LL_DMA_CHANNEL_ALL	DMA Channel all (used only for function

Clear Flags Defines

LL_DMA_IFCR_CGIF1	Channel 1 global flag
LL_DMA_IFCR_CTCIF1	Channel 1 transfer complete flag
LL_DMA_IFCR_CHTIF1	Channel 1 half transfer flag
LL_DMA_IFCR_CTEIF1	Channel 1 transfer error flag
LL_DMA_IFCR_CGIF2	Channel 2 global flag
LL_DMA_IFCR_CTCIF2	Channel 2 transfer complete flag
LL_DMA_IFCR_CHTIF2	Channel 2 half transfer flag
LL_DMA_IFCR_CTEIF2	Channel 2 transfer error flag
LL_DMA_IFCR_CGIF3	Channel 3 global flag
LL_DMA_IFCR_CTCIF3	Channel 3 transfer complete flag
LL_DMA_IFCR_CHTIF3	Channel 3 half transfer flag
LL_DMA_IFCR_CTEIF3	Channel 3 transfer error flag
LL_DMA_IFCR_CGIF4	Channel 4 global flag
LL_DMA_IFCR_CTCIF4	Channel 4 transfer complete flag
LL_DMA_IFCR_CHTIF4	Channel 4 half transfer flag
LL_DMA_IFCR_CTEIF4	Channel 4 transfer error flag
LL_DMA_IFCR_CGIF5	Channel 5 global flag
LL_DMA_IFCR_CTCIF5	Channel 5 transfer complete flag
LL_DMA_IFCR_CHTIF5	Channel 5 half transfer flag
LL_DMA_IFCR_CTEIF5	Channel 5 transfer error flag
LL_DMA_IFCR_CGIF6	Channel 6 global flag
LL_DMA_IFCR_CTCIF6	Channel 6 transfer complete flag
LL_DMA_IFCR_CHTIF6	Channel 6 half transfer flag
LL_DMA_IFCR_CTEIF6	Channel 6 transfer error flag
LL_DMA_IFCR_CGIF7	Channel 7 global flag
LL_DMA_IFCR_CTCIF7	Channel 7 transfer complete flag
LL_DMA_IFCR_CHTIF7	Channel 7 half transfer flag
LL_DMA_IFCR_CTEIF7	Channel 7 transfer error flag

Transfer Direction

LL_DMA_DIRECTION_PERIPH_TO_MEMORY	Peripheral to memory direction
LL_DMA_DIRECTION_MEMORY_TO_PERIPH	Memory to peripheral direction
LL_DMA_DIRECTION_MEMORY_TO_MEMORY	Memory to memory direction

Get Flags Defines

LL_DMA_ISR_GIF1	Channel 1 global flag
LL_DMA_ISR_TCIF1	Channel 1 transfer complete flag
LL_DMA_ISR_HTIF1	Channel 1 half transfer flag
LL_DMA_ISR_TEIF1	Channel 1 transfer error flag
LL_DMA_ISR_GIF2	Channel 2 global flag
LL_DMA_ISR_TCIF2	Channel 2 transfer complete flag
LL_DMA_ISR_HTIF2	Channel 2 half transfer flag
LL_DMA_ISR_TEIF2	Channel 2 transfer error flag
LL_DMA_ISR_GIF3	Channel 3 global flag
LL_DMA_ISR_TCIF3	Channel 3 transfer complete flag
LL_DMA_ISR_HTIF3	Channel 3 half transfer flag
LL_DMA_ISR_TEIF3	Channel 3 transfer error flag
LL_DMA_ISR_GIF4	Channel 4 global flag
LL_DMA_ISR_TCIF4	Channel 4 transfer complete flag
LL_DMA_ISR_HTIF4	Channel 4 half transfer flag
LL_DMA_ISR_TEIF4	Channel 4 transfer error flag
LL_DMA_ISR_GIF5	Channel 5 global flag
LL_DMA_ISR_TCIF5	Channel 5 transfer complete flag
LL_DMA_ISR_HTIF5	Channel 5 half transfer flag
LL_DMA_ISR_TEIF5	Channel 5 transfer error flag
LL_DMA_ISR_GIF6	Channel 6 global flag
LL_DMA_ISR_TCIF6	Channel 6 transfer complete flag
LL_DMA_ISR_HTIF6	Channel 6 half transfer flag
LL_DMA_ISR_TEIF6	Channel 6 transfer error flag
LL_DMA_ISR_GIF7	Channel 7 global flag
LL_DMA_ISR_TCIF7	Channel 7 transfer complete flag
LL_DMA_ISR_HTIF7	Channel 7 half transfer flag
LL_DMA_ISR_TEIF7	Channel 7 transfer error flag

IT Defines

LL_DMA_CCR_TCIE	Transfer complete interrupt
LL_DMA_CCR_HTIE	Half Transfer interrupt
LL_DMA_CCR_TEIE	Transfer error interrupt

Memory data alignment

LL_DMA_MDATAALIGN_BYTE	Memory data alignment: Byte
LL_DMA_MDATAALIGN_HALFWORD	Memory data alignment: HalfWord
LL_DMA_MDATAALIGN_WORD	Memory data alignment: Word

Memory increment mode

LL_DMA_MEMORY_INCREMENT	Memory increment mode Enable
LL_DMA_MEMORY_NOINCREMENT	Memory increment mode Disable

Transfer mode

LL_DMA_MODE_NORMAL	Normal Mode
LL_DMA_MODE_CIRCULAR	Circular Mode

Peripheral data alignment

LL_DMA_PDATAALIGN_BYTE	Peripheral data alignment: Byte
LL_DMA_PDATAALIGN_HALFWORD	Peripheral data alignment: HalfWord
LL_DMA_PDATAALIGN_WORD	Peripheral data alignment: Word

Peripheral increment mode

LL_DMA_PERIPH_INCREMENT	Peripheral increment mode Enable
LL_DMA_PERIPH_NOINCREMENT	Peripheral increment mode Disable

Transfer Priority level

LL_DMA_PRIORITY_LOW	Priority level: Low
LL_DMA_PRIORITY_MEDIUM	Priority level: Medium
LL_DMA_PRIORITY_HIGH	Priority level: High
LL_DMA_PRIORITY_VERYHIGH	Priority level: Very_High

Convert DMAxChannely

__LL_DMA_GET_INSTANCE	**Description:** • Convert DMAx_Channely into DMAx. **Parameters:** • __CHANNEL_INSTANCE__: DMAx_Channely **Return value:** • DMAx
__LL_DMA_GET_CHANNEL	**Description:** • Convert DMAx_Channely into LL_DMA_CHANNEL_y. **Parameters:** • __CHANNEL_INSTANCE__: DMAx_Channely **Return value:** • LL_DMA_CHANNEL_y

__LL_DMA_GET_CHANNEL_INSTANCE

Description:

- Convert DMA Instance DMAx and LL_DMA_CHANNEL_y into DMAx_Channely.

Parameters:

- __DMA_INSTANCE__: DMAx
- __CHANNEL__: LL_DMA_CHANNEL_y

Return value:

- DMAx_Channely

Common Write and read registers macros

LL_DMA_WriteReg

Description:

- Write a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMA_ReadReg

Description:

- Read a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance
- __REG__: Register to be read

Return value:

- Register: value

83 LL EXTI Generic Driver

83.1 EXTI Firmware driver registers structures

83.1.1 LL_EXTI_InitTypeDef

Data Fields

- ***uint32_t Line_0_31***
- ***uint32_t Line_32_63***
- ***FunctionalState LineCommand***
- ***uint8_t Mode***
- ***uint8_t Trigger***

Field Documentation

- ***uint32_t LL_EXTI_InitTypeDef::Line_0_31***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 0 to 31 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***uint32_t LL_EXTI_InitTypeDef::Line_32_63***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 32 to 63 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***FunctionalState LL_EXTI_InitTypeDef::LineCommand***
  Specifies the new state of the selected EXTI lines. This parameter can be set either to ENABLE or DISABLE
- ***uint8_t LL_EXTI_InitTypeDef::Mode***
  Specifies the mode for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_MODE***.
- ***uint8_t LL_EXTI_InitTypeDef::Trigger***
  Specifies the trigger signal active edge for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_TRIGGER***.

83.2 EXTI Firmware driver API description

83.2.1 Detailed description of functions

LL_EXTI_EnableIT_0_31

Function name	**__STATIC_INLINE void LL_EXTI_EnableIT_0_31 (uint32_t ExtiLine)**
Function description	Enable ExtiLine Interrupt request for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be one of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9

- LL_EXTI_LINE_10
- LL_EXTI_LINE_11
- LL_EXTI_LINE_12
- LL_EXTI_LINE_13
- LL_EXTI_LINE_14
- LL_EXTI_LINE_15
- LL_EXTI_LINE_16
- LL_EXTI_LINE_17
- LL_EXTI_LINE_18
- LL_EXTI_LINE_19
- LL_EXTI_LINE_20
- LL_EXTI_LINE_21
- LL_EXTI_LINE_22
- LL_EXTI_LINE_23
- LL_EXTI_LINE_24
- LL_EXTI_LINE_25
- LL_EXTI_LINE_26
- LL_EXTI_LINE_27
- LL_EXTI_LINE_28
- LL_EXTI_LINE_29
- LL_EXTI_LINE_30
- LL_EXTI_LINE_31
- LL_EXTI_LINE_ALL_0_31

Return values	• **None:**
Notes	• The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• IMR1 IMx LL_EXTI_EnableIT_0_31

LL_EXTI_EnableIT_32_63

Function name	**__STATIC_INLINE void LL_EXTI_EnableIT_32_63 (uint32_t ExtiLine)**
Function description	Enable ExtiLine Interrupt request for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be one of the following values: – LL_EXTI_LINE_32 – LL_EXTI_LINE_33 – LL_EXTI_LINE_34(*) – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38 – LL_EXTI_LINE_39(*) – LL_EXTI_LINE_40(*) – LL_EXTI_LINE_ALL_32_63

Return values	• **None:**
Notes	• The reset value for the direct lines (lines from 32 to 34, line 39) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on. • (*): Available in some devices
Reference Manual to LL API cross reference:	• IMR2 IMx LL_EXTI_EnableIT_32_63

LL_EXTI_DisableIT_0_31

Function name	**__STATIC_INLINE void LL_EXTI_DisableIT_0_31 (uint32_t ExtiLine)**
Function description	Disable ExtiLine Interrupt request for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be one of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_17 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19 – LL_EXTI_LINE_20 – LL_EXTI_LINE_21 – LL_EXTI_LINE_22 – LL_EXTI_LINE_23 – LL_EXTI_LINE_24 – LL_EXTI_LINE_25 – LL_EXTI_LINE_26 – LL_EXTI_LINE_27 – LL_EXTI_LINE_28 – LL_EXTI_LINE_29 – LL_EXTI_LINE_30 – LL_EXTI_LINE_31 – LL_EXTI_LINE_ALL_0_31
Return values	• **None:**

Notes	• The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• IMR1 IMx LL_EXTI_DisableIT_0_31

LL_EXTI_DisableIT_32_63

Function name	**__STATIC_INLINE void LL_EXTI_DisableIT_32_63 (uint32_t ExtiLine)**
Function description	Disable ExtiLine Interrupt request for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be one of the following values: – LL_EXTI_LINE_32 – LL_EXTI_LINE_33 – LL_EXTI_LINE_34(*) – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38 – LL_EXTI_LINE_39(*) – LL_EXTI_LINE_40(*) – LL_EXTI_LINE_ALL_32_63
Return values	• **None:**
Notes	• The reset value for the direct lines (lines from 32 to 34, line 39) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on. • (*): Available in some devices
Reference Manual to LL API cross reference:	• IMR2 IMx LL_EXTI_DisableIT_32_63

LL_EXTI_IsEnabledIT_0_31

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_0_31 (uint32_t ExtiLine)**
Function description	Indicate if ExtiLine Interrupt request is enabled for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be one of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7

– LL_EXTI_LINE_8
– LL_EXTI_LINE_9
– LL_EXTI_LINE_10
– LL_EXTI_LINE_11
– LL_EXTI_LINE_12
– LL_EXTI_LINE_13
– LL_EXTI_LINE_14
– LL_EXTI_LINE_15
– LL_EXTI_LINE_16
– LL_EXTI_LINE_17
– LL_EXTI_LINE_18
– LL_EXTI_LINE_19
– LL_EXTI_LINE_20
– LL_EXTI_LINE_21
– LL_EXTI_LINE_22
– LL_EXTI_LINE_23
– LL_EXTI_LINE_24
– LL_EXTI_LINE_25
– LL_EXTI_LINE_26
– LL_EXTI_LINE_27
– LL_EXTI_LINE_28
– LL_EXTI_LINE_29
– LL_EXTI_LINE_30
– LL_EXTI_LINE_31
– LL_EXTI_LINE_ALL_0_31

Return values
- **State:** of bit (1 or 0).

Notes
- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- IMR1 IMx LL_EXTI_IsEnabledIT_0_31

LL_EXTI_IsEnabledIT_32_63

Function name
__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_32_63 (uint32_t ExtiLine)

Function description
Indicate if ExtiLine Interrupt request is enabled for Lines in range 32 to 63.

Parameters
- **ExtiLine:** This parameter can be one of the following values:
  – LL_EXTI_LINE_32
  – LL_EXTI_LINE_33
  – LL_EXTI_LINE_34(*)
  – LL_EXTI_LINE_35
  – LL_EXTI_LINE_36
  – LL_EXTI_LINE_37
  – LL_EXTI_LINE_38
  – LL_EXTI_LINE_39(*)

	– LL_EXTI_LINE_40(*) – LL_EXTI_LINE_ALL_32_63
Return values	• **State:** of bit (1 or 0).
Notes	• The reset value for the direct lines (lines from 32 to 34, line 39) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on. • (*): Available in some devices
Reference Manual to LL API cross reference:	• IMR2 IMx LL_EXTI_IsEnabledIT_32_63

LL_EXTI_EnableEvent_0_31

Function name	**__STATIC_INLINE void LL_EXTI_EnableEvent_0_31 (uint32_t ExtiLine)**
Function description	Enable ExtiLine Event request for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be one of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_17 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19 – LL_EXTI_LINE_20 – LL_EXTI_LINE_21 – LL_EXTI_LINE_22 – LL_EXTI_LINE_23 – LL_EXTI_LINE_24 – LL_EXTI_LINE_25 – LL_EXTI_LINE_26 – LL_EXTI_LINE_27 – LL_EXTI_LINE_28 – LL_EXTI_LINE_29 – LL_EXTI_LINE_30 – LL_EXTI_LINE_31 – LL_EXTI_LINE_ALL_0_31

Return values	• **None:**
Notes	• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• EMR1 EMx LL_EXTI_EnableEvent_0_31

LL_EXTI_EnableEvent_32_63

Function name	**__STATIC_INLINE void LL_EXTI_EnableEvent_32_63 (uint32_t ExtiLine)**
Function description	Enable ExtiLine Event request for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_32 – LL_EXTI_LINE_33 – LL_EXTI_LINE_34(*) – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38 – LL_EXTI_LINE_39(*) – LL_EXTI_LINE_40(*) – LL_EXTI_LINE_ALL_32_63
Return values	• **None:**
Notes	• (*): Available in some devices
Reference Manual to LL API cross reference:	• EMR2 EMx LL_EXTI_EnableEvent_32_63

LL_EXTI_DisableEvent_0_31

Function name	**__STATIC_INLINE void LL_EXTI_DisableEvent_0_31 (uint32_t ExtiLine)**
Function description	Disable ExtiLine Event request for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be one of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12

		– LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_17 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19 – LL_EXTI_LINE_20 – LL_EXTI_LINE_21 – LL_EXTI_LINE_22 – LL_EXTI_LINE_23 – LL_EXTI_LINE_24 – LL_EXTI_LINE_25 – LL_EXTI_LINE_26 – LL_EXTI_LINE_27 – LL_EXTI_LINE_28 – LL_EXTI_LINE_29 – LL_EXTI_LINE_30 – LL_EXTI_LINE_31 – LL_EXTI_LINE_ALL_0_31
Return values	•	**None:**
Notes	•	Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	•	EMR1 EMx LL_EXTI_DisableEvent_0_31

LL_EXTI_DisableEvent_32_63

Function name	**__STATIC_INLINE void LL_EXTI_DisableEvent_32_63 (uint32_t ExtiLine)**
Function description	Disable ExtiLine Event request for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_32 – LL_EXTI_LINE_33 – LL_EXTI_LINE_34(*) – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38 – LL_EXTI_LINE_39(*) – LL_EXTI_LINE_40(*) – LL_EXTI_LINE_ALL_32_63
Return values	• **None:**
Notes	• (*): Available in some devices
Reference Manual to LL API cross	• EMR2 EMx LL_EXTI_DisableEvent_32_63

reference:

LL_EXTI_IsEnabledEvent_0_31

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_0_31 (uint32_t ExtiLine)**
Function description	Indicate if ExtiLine Event request is enabled for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be one of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_17 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19 – LL_EXTI_LINE_20 – LL_EXTI_LINE_21 – LL_EXTI_LINE_22 – LL_EXTI_LINE_23 – LL_EXTI_LINE_24 – LL_EXTI_LINE_25 – LL_EXTI_LINE_26 – LL_EXTI_LINE_27 – LL_EXTI_LINE_28 – LL_EXTI_LINE_29 – LL_EXTI_LINE_30 – LL_EXTI_LINE_31 – LL_EXTI_LINE_ALL_0_31
Return values	• **State:** of bit (1 or 0).
Notes	• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• EMR1 EMx LL_EXTI_IsEnabledEvent_0_31

LL_EXTI_IsEnabledEvent_32_63

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_32_63 (uint32_t ExtiLine)**
Function description	Indicate if ExtiLine Event request is enabled for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_32 – LL_EXTI_LINE_33 – LL_EXTI_LINE_34(*) – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38 – LL_EXTI_LINE_39(*) – LL_EXTI_LINE_40(*) – LL_EXTI_LINE_ALL_32_63
Return values	• **State:** of bit (1 or 0).
Notes	• (*): Available in some devices
Reference Manual to LL API cross reference:	• EMR2 EMx LL_EXTI_IsEnabledEvent_32_63

LL_EXTI_EnableRisingTrig_0_31

Function name	**__STATIC_INLINE void LL_EXTI_EnableRisingTrig_0_31 (uint32_t ExtiLine)**
Function description	Enable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19

- LL_EXTI_LINE_20
- LL_EXTI_LINE_21
- LL_EXTI_LINE_22
- LL_EXTI_LINE_29
- LL_EXTI_LINE_30
- LL_EXTI_LINE_31

Return values
- **None:**

Notes
- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- RTSR1 RTx LL_EXTI_EnableRisingTrig_0_31

LL_EXTI_EnableRisingTrig_32_63

Function name
__STATIC_INLINE void LL_EXTI_EnableRisingTrig_32_63 (uint32_t ExtiLine)

Function description
Enable ExtiLine Rising Edge Trigger for Lines in range 32 to 63.

Parameters
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38

Return values
- **None:**

Notes
- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set.Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:
- RTSR2 RTx LL_EXTI_EnableRisingTrig_32_63

LL_EXTI_DisableRisingTrig_0_31

Function name
__STATIC_INLINE void LL_EXTI_DisableRisingTrig_0_31 (uint32_t ExtiLine)

Function description
Disable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0

- – LL_EXTI_LINE_1
- – LL_EXTI_LINE_2
- – LL_EXTI_LINE_3
- – LL_EXTI_LINE_4
- – LL_EXTI_LINE_5
- – LL_EXTI_LINE_6
- – LL_EXTI_LINE_7
- – LL_EXTI_LINE_8
- – LL_EXTI_LINE_9
- – LL_EXTI_LINE_10
- – LL_EXTI_LINE_11
- – LL_EXTI_LINE_12
- – LL_EXTI_LINE_13
- – LL_EXTI_LINE_14
- – LL_EXTI_LINE_15
- – LL_EXTI_LINE_16
- – LL_EXTI_LINE_18
- – LL_EXTI_LINE_19
- – LL_EXTI_LINE_20
- – LL_EXTI_LINE_21
- – LL_EXTI_LINE_22
- – LL_EXTI_LINE_29
- – LL_EXTI_LINE_30
- – LL_EXTI_LINE_31

Return values	• **None:**
Notes	• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• RTSR1 RTx LL_EXTI_DisableRisingTrig_0_31

LL_EXTI_DisableRisingTrig_32_63

Function name	**__STATIC_INLINE void LL_EXTI_DisableRisingTrig_32_63 (uint32_t ExtiLine)**
Function description	Disable ExtiLine Rising Edge Trigger for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38
Return values	• **None:**

Notes	• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
Reference Manual to LL API cross reference:	• RTSR2 RTx LL_EXTI_DisableRisingTrig_32_63

LL_EXTI_IsEnabledRisingTrig_0_31

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_0_31 (uint32_t ExtiLine)**
Function description	Check if rising edge trigger is enabled for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19 – LL_EXTI_LINE_20 – LL_EXTI_LINE_21 – LL_EXTI_LINE_22 – LL_EXTI_LINE_29 – LL_EXTI_LINE_30 – LL_EXTI_LINE_31
Return values	• **State:** of bit (1 or 0).
Notes	• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• RTSR1 RTx LL_EXTI_IsEnabledRisingTrig_0_31

LL_EXTI_IsEnabledRisingTrig_32_63

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_32_63 (uint32_t ExtiLine)**
Function description	Check if rising edge trigger is enabled for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• RTSR2 RTx LL_EXTI_IsEnabledRisingTrig_32_63

LL_EXTI_EnableFallingTrig_0_31

Function name	**__STATIC_INLINE void LL_EXTI_EnableFallingTrig_0_31 (uint32_t ExtiLine)**
Function description	Enable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19 – LL_EXTI_LINE_20 – LL_EXTI_LINE_21 – LL_EXTI_LINE_22 – LL_EXTI_LINE_29 – LL_EXTI_LINE_30 – LL_EXTI_LINE_31
Return values	• **None:**

Notes	• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• FTSR1 FTx LL_EXTI_EnableFallingTrig_0_31

LL_EXTI_EnableFallingTrig_32_63

Function name	**__STATIC_INLINE void LL_EXTI_EnableFallingTrig_32_63 (uint32_t ExtiLine)**
Function description	Enable ExtiLine Falling Edge Trigger for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38
Return values	• **None:**
Notes	• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
Reference Manual to LL API cross reference:	• FTSR2 FTx LL_EXTI_EnableFallingTrig_32_63

LL_EXTI_DisableFallingTrig_0_31

Function name	**__STATIC_INLINE void LL_EXTI_DisableFallingTrig_0_31 (uint32_t ExtiLine)**
Function description	Disable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8

- LL_EXTI_LINE_9
- LL_EXTI_LINE_10
- LL_EXTI_LINE_11
- LL_EXTI_LINE_12
- LL_EXTI_LINE_13
- LL_EXTI_LINE_14
- LL_EXTI_LINE_15
- LL_EXTI_LINE_16
- LL_EXTI_LINE_18
- LL_EXTI_LINE_19
- LL_EXTI_LINE_20
- LL_EXTI_LINE_21
- LL_EXTI_LINE_22
- LL_EXTI_LINE_29
- LL_EXTI_LINE_30
- LL_EXTI_LINE_31

Return values
- **None:**

Notes
- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- FTSR1 FTx LL_EXTI_DisableFallingTrig_0_31

LL_EXTI_DisableFallingTrig_32_63

Function name
__STATIC_INLINE void LL_EXTI_DisableFallingTrig_32_63 (uint32_t ExtiLine)

Function description
Disable ExtiLine Falling Edge Trigger for Lines in range 32 to 63.

Parameters
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38

Return values
- **None:**

Notes
- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross
- FTSR2 FTx LL_EXTI_DisableFallingTrig_32_63

reference:

LL_EXTI_IsEnabledFallingTrig_0_31

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_0_31 (uint32_t ExtiLine)**
Function description	Check if falling edge trigger is enabled for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19 – LL_EXTI_LINE_20 – LL_EXTI_LINE_21 – LL_EXTI_LINE_22 – LL_EXTI_LINE_29 – LL_EXTI_LINE_30 – LL_EXTI_LINE_31
Return values	• **State:** of bit (1 or 0).
Notes	• Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• FTSR1 FTx LL_EXTI_IsEnabledFallingTrig_0_31

LL_EXTI_IsEnabledFallingTrig_32_63

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_32_63 (uint32_t ExtiLine)**
Function description	Check if falling edge trigger is enabled for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_35 – LL_EXTI_LINE_36

		– LL_EXTI_LINE_37 – LL_EXTI_LINE_38
Return values	•	**State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	•	FTSR2 FTx LL_EXTI_IsEnabledFallingTrig_32_63

LL_EXTI_GenerateSWI_0_31

Function name	**__STATIC_INLINE void LL_EXTI_GenerateSWI_0_31 (uint32_t ExtiLine)**
Function description	Generate a software Interrupt Event for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19 – LL_EXTI_LINE_20 – LL_EXTI_LINE_21 – LL_EXTI_LINE_22 – LL_EXTI_LINE_29 – LL_EXTI_LINE_30 – LL_EXTI_LINE_31
Return values	• **None:**
Notes	• If the interrupt is enabled on this line in the EXTI_IMR1, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR1 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR1 register (by writing a 1 into the bit) • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross	• SWIER1 SWIx LL_EXTI_GenerateSWI_0_31

reference:

LL_EXTI_GenerateSWI_32_63

Function name	**__STATIC_INLINE void LL_EXTI_GenerateSWI_32_63 (uint32_t ExtiLine)**
Function description	Generate a software Interrupt Event for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38
Return values	• **None:**
Notes	• If the interrupt is enabled on this line inthe EXTI_IMR2, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR2 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR2 register (by writing a 1 into the bit)
Reference Manual to LL API cross reference:	• SWIER2 SWIx LL_EXTI_GenerateSWI_32_63

LL_EXTI_IsActiveFlag_0_31

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsActiveFlag_0_31 (uint32_t ExtiLine)**
Function description	Check if the ExtLine Flag is set or not for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19 – LL_EXTI_LINE_20

	– LL_EXTI_LINE_21 – LL_EXTI_LINE_22 – LL_EXTI_LINE_29 – LL_EXTI_LINE_30 – LL_EXTI_LINE_31
Return values	• **State:** of bit (1 or 0).
Notes	• This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• PR1 PIFx LL_EXTI_IsActiveFlag_0_31

LL_EXTI_IsActiveFlag_32_63

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsActiveFlag_32_63 (uint32_t ExtiLine)**
Function description	Check if the ExtLine Flag is set or not for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38
Return values	• **State:** of bit (1 or 0).
Notes	• This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
Reference Manual to LL API cross reference:	• PR2 PIFx LL_EXTI_IsActiveFlag_32_63

LL_EXTI_ReadFlag_0_31

Function name	**__STATIC_INLINE uint32_t LL_EXTI_ReadFlag_0_31 (uint32_t ExtiLine)**
Function description	Read ExtLine Combination Flag for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8

– LL_EXTI_LINE_9
– LL_EXTI_LINE_10
– LL_EXTI_LINE_11
– LL_EXTI_LINE_12
– LL_EXTI_LINE_13
– LL_EXTI_LINE_14
– LL_EXTI_LINE_15
– LL_EXTI_LINE_16
– LL_EXTI_LINE_18
– LL_EXTI_LINE_19
– LL_EXTI_LINE_20
– LL_EXTI_LINE_21
– LL_EXTI_LINE_22
– LL_EXTI_LINE_29
– LL_EXTI_LINE_30
– LL_EXTI_LINE_31

Return values
- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes
- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- PR1 PIFx LL_EXTI_ReadFlag_0_31

LL_EXTI_ReadFlag_32_63

Function name
__STATIC_INLINE uint32_t LL_EXTI_ReadFlag_32_63 (uint32_t ExtiLine)

Function description
Read ExtLine Combination Flag for Lines in range 32 to 63.

Parameters
- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_35
  – LL_EXTI_LINE_36
  – LL_EXTI_LINE_37
  – LL_EXTI_LINE_38

Return values
- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes
- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:
- PR2 PIFx LL_EXTI_ReadFlag_32_63

LL_EXTI_ClearFlag_0_31

Function name
__STATIC_INLINE void LL_EXTI_ClearFlag_0_31 (uint32_t ExtiLine)

Function description	Clear ExtLine Flags for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19 – LL_EXTI_LINE_20 – LL_EXTI_LINE_21 – LL_EXTI_LINE_22 – LL_EXTI_LINE_29 – LL_EXTI_LINE_30 – LL_EXTI_LINE_31
Return values	• **None:**
Notes	• This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• PR1 PIFx LL_EXTI_ClearFlag_0_31

LL_EXTI_ClearFlag_32_63

Function name	**__STATIC_INLINE void LL_EXTI_ClearFlag_32_63 (uint32_t ExtiLine)**
Function description	Clear ExtLine Flags for Lines in range 32 to 63.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_35 – LL_EXTI_LINE_36 – LL_EXTI_LINE_37 – LL_EXTI_LINE_38
Return values	• **None:**

Notes	• This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
Reference Manual to LL API cross reference:	• PR2 PIFx LL_EXTI_ClearFlag_32_63

LL_EXTI_Init

Function name	**uint32_t LL_EXTI_Init (LL_EXTI_InitTypeDef * EXTI_InitStruct)**
Function description	Initialize the EXTI registers according to the specified parameters in EXTI_InitStruct.
Parameters	• **EXTI_InitStruct:** pointer to a LL_EXTI_InitTypeDef structure.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: EXTI registers are initialized – ERROR: not applicable

LL_EXTI_DeInit

Function name	**uint32_t LL_EXTI_DeInit (void)**
Function description	De-initialize the EXTI registers to their default reset values.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: EXTI registers are de-initialized – ERROR: not applicable

LL_EXTI_StructInit

Function name	**void LL_EXTI_StructInit (LL_EXTI_InitTypeDef * EXTI_InitStruct)**
Function description	Set each LL_EXTI_InitTypeDef field to default value.
Parameters	• **EXTI_InitStruct:** Pointer to a LL_EXTI_InitTypeDef structure.
Return values	• **None:**

83.3 EXTI Firmware driver defines

83.3.1 EXTI

LINE

LL_EXTI_LINE_0	Extended line 0
LL_EXTI_LINE_1	Extended line 1
LL_EXTI_LINE_2	Extended line 2
LL_EXTI_LINE_3	Extended line 3
LL_EXTI_LINE_4	Extended line 4
LL_EXTI_LINE_5	Extended line 5
LL_EXTI_LINE_6	Extended line 6

LL_EXTI_LINE_7	Extended line 7
LL_EXTI_LINE_8	Extended line 8
LL_EXTI_LINE_9	Extended line 9
LL_EXTI_LINE_10	Extended line 10
LL_EXTI_LINE_11	Extended line 11
LL_EXTI_LINE_12	Extended line 12
LL_EXTI_LINE_13	Extended line 13
LL_EXTI_LINE_14	Extended line 14
LL_EXTI_LINE_15	Extended line 15
LL_EXTI_LINE_16	Extended line 16
LL_EXTI_LINE_17	Extended line 17
LL_EXTI_LINE_18	Extended line 18
LL_EXTI_LINE_19	Extended line 19
LL_EXTI_LINE_20	Extended line 20
LL_EXTI_LINE_21	Extended line 21
LL_EXTI_LINE_22	Extended line 22
LL_EXTI_LINE_23	Extended line 23
LL_EXTI_LINE_24	Extended line 24
LL_EXTI_LINE_25	Extended line 25
LL_EXTI_LINE_26	Extended line 26
LL_EXTI_LINE_27	Extended line 27
LL_EXTI_LINE_28	Extended line 28
LL_EXTI_LINE_29	Extended line 29
LL_EXTI_LINE_30	Extended line 30
LL_EXTI_LINE_31	Extended line 31
LL_EXTI_LINE_ALL_0_31	All Extended line not reserved
LL_EXTI_LINE_32	Extended line 32
LL_EXTI_LINE_33	Extended line 33
LL_EXTI_LINE_35	Extended line 35
LL_EXTI_LINE_36	Extended line 36
LL_EXTI_LINE_37	Extended line 37
LL_EXTI_LINE_38	Extended line 38
LL_EXTI_LINE_40	Extended line 40
LL_EXTI_LINE_ALL_32_63	All Extended line not reserved
LL_EXTI_LINE_ALL	All Extended line
LL_EXTI_LINE_NONE	None Extended line

Mode

LL_EXTI_MODE_IT	Interrupt Mode
LL_EXTI_MODE_EVENT	Event Mode
LL_EXTI_MODE_IT_EVENT	Interrupt & Event Mode

Edge Trigger

LL_EXTI_TRIGGER_NONE	No Trigger Mode
LL_EXTI_TRIGGER_RISING	Trigger Rising Mode
LL_EXTI_TRIGGER_FALLING	Trigger Falling Mode
LL_EXTI_TRIGGER_RISING_FALLING	Trigger Rising & Falling Mode

Common Write and read registers Macros

LL_EXTI_WriteReg

Description:

- Write a value in EXTI register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_EXTI_ReadReg

Description:

- Read a value in EXTI register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

84 LL GPIO Generic Driver

84.1 GPIO Firmware driver registers structures

84.1.1 LL_GPIO_InitTypeDef

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Speed***
- ***uint32_t OutputType***
- ***uint32_t Pull***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t LL_GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_LL_EC_PIN***
- ***uint32_t LL_GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_LL_EC_MODE***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinMode()**.
- ***uint32_t LL_GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_LL_EC_SPEED***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinSpeed()**.
- ***uint32_t LL_GPIO_InitTypeDef::OutputType***
  Specifies the operating output type for the selected pins. This parameter can be a value of ***GPIO_LL_EC_OUTPUT***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinOutputType()**.
- ***uint32_t LL_GPIO_InitTypeDef::Pull***
  Specifies the operating Pull-up/Pull down for the selected pins. This parameter can be a value of ***GPIO_LL_EC_PULL***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinPull()**.
- ***uint32_t LL_GPIO_InitTypeDef::Alternate***
  Specifies the Peripheral to be connected to the selected pins. This parameter can be a value of ***GPIO_LL_EC_AF***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetAFPin_0_7()** and **LL_GPIO_SetAFPin_8_15()**.

84.2 GPIO Firmware driver API description

84.2.1 Detailed description of functions

LL_GPIO_SetPinMode

Function name	**__STATIC_INLINE void LL_GPIO_SetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Mode)**
Function description	Configure gpio mode for a dedicated pin on dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values:

- LL_GPIO_PIN_0
- LL_GPIO_PIN_1
- LL_GPIO_PIN_2
- LL_GPIO_PIN_3
- LL_GPIO_PIN_4
- LL_GPIO_PIN_5
- LL_GPIO_PIN_6
- LL_GPIO_PIN_7
- LL_GPIO_PIN_8
- LL_GPIO_PIN_9
- LL_GPIO_PIN_10
- LL_GPIO_PIN_11
- LL_GPIO_PIN_12
- LL_GPIO_PIN_13
- LL_GPIO_PIN_14
- LL_GPIO_PIN_15

- **Mode:** This parameter can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Return values
- **None:**

Notes
- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:
- MODER MODEy LL_GPIO_SetPinMode

LL_GPIO_GetPinMode

Function name
__STATIC_INLINE uint32_t LL_GPIO_GetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description
Return gpio mode for a dedicated pin on dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13

– LL_GPIO_PIN_14
– LL_GPIO_PIN_15

Return values	• **Returned:** value can be one of the following values: – LL_GPIO_MODE_INPUT – LL_GPIO_MODE_OUTPUT – LL_GPIO_MODE_ALTERNATE – LL_GPIO_MODE_ANALOG
Notes	• I/O mode can be Input mode, General purpose output, Alternate function mode or Analog. • Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• MODER MODEy LL_GPIO_GetPinMode

LL_GPIO_SetPinOutputType

Function name	**__STATIC_INLINE void LL_GPIO_SetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t PinMask, uint32_t OutputType)**
Function description	Configure gpio output type for several pins on dedicated port.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL • **OutputType:** This parameter can be one of the following values: – LL_GPIO_OUTPUT_PUSHPULL – LL_GPIO_OUTPUT_OPENDRAIN
Return values	• **None:**
Notes	• Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.
Reference Manual to LL API cross	• OTYPER OTy LL_GPIO_SetPinOutputType

reference:

LL_GPIO_GetPinOutputType

Function name	**__STATIC_INLINE uint32_t LL_GPIO_GetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t Pin)**
Function description	Return gpio output type for several pins on dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL
Return values	• **Returned:** value can be one of the following values: – LL_GPIO_OUTPUT_PUSHPULL – LL_GPIO_OUTPUT_OPENDRAIN
Notes	• Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain. • Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• OTYPER OTy LL_GPIO_GetPinOutputType

LL_GPIO_SetPinSpeed

Function name	**__STATIC_INLINE void LL_GPIO_SetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Speed)**
Function description	Configure gpio speed for a dedicated pin on dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5

- – LL_GPIO_PIN_6
- – LL_GPIO_PIN_7
- – LL_GPIO_PIN_8
- – LL_GPIO_PIN_9
- – LL_GPIO_PIN_10
- – LL_GPIO_PIN_11
- – LL_GPIO_PIN_12
- – LL_GPIO_PIN_13
- – LL_GPIO_PIN_14
- – LL_GPIO_PIN_15
- **Speed:** This parameter can be one of the following values:
  - – LL_GPIO_SPEED_FREQ_LOW
  - – LL_GPIO_SPEED_FREQ_MEDIUM
  - – LL_GPIO_SPEED_FREQ_HIGH
  - – LL_GPIO_SPEED_FREQ_VERY_HIGH

Return values
- **None:**

Notes
- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:
- OSPEEDR OSPEEDy LL_GPIO_SetPinSpeed

LL_GPIO_GetPinSpeed

Function name
__STATIC_INLINE uint32_t LL_GPIO_GetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description
Return gpio speed for a dedicated pin on dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - – LL_GPIO_PIN_0
  - – LL_GPIO_PIN_1
  - – LL_GPIO_PIN_2
  - – LL_GPIO_PIN_3
  - – LL_GPIO_PIN_4
  - – LL_GPIO_PIN_5
  - – LL_GPIO_PIN_6
  - – LL_GPIO_PIN_7
  - – LL_GPIO_PIN_8
  - – LL_GPIO_PIN_9
  - – LL_GPIO_PIN_10
  - – LL_GPIO_PIN_11
  - – LL_GPIO_PIN_12
  - – LL_GPIO_PIN_13
  - – LL_GPIO_PIN_14
  - – LL_GPIO_PIN_15

Return values
- **Returned:** value can be one of the following values:
  - – LL_GPIO_SPEED_FREQ_LOW
  - – LL_GPIO_SPEED_FREQ_MEDIUM

	– LL_GPIO_SPEED_FREQ_HIGH – LL_GPIO_SPEED_FREQ_VERY_HIGH
Notes	• I/O speed can be Low, Medium, Fast or High speed. • Warning: only one pin can be passed as parameter. • Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.
Reference Manual to LL API cross reference:	• OSPEEDR OSPEEDy LL_GPIO_GetPinSpeed

LL_GPIO_SetPinPull

Function name	**__STATIC_INLINE void LL_GPIO_SetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Pull)**
Function description	Configure gpio pull-up or pull-down for a dedicated pin on a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 • **Pull:** This parameter can be one of the following values: – LL_GPIO_PULL_NO – LL_GPIO_PULL_UP – LL_GPIO_PULL_DOWN
Return values	• **None:**
Notes	• Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• PUPDR PUPDy LL_GPIO_SetPinPull

LL_GPIO_GetPinPull

Function name	**__STATIC_INLINE uint32_t LL_GPIO_GetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin)**
Function description	Return gpio pull-up or pull-down for a dedicated pin on a dedicated

port.

Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15
Return values	• **Returned:** value can be one of the following values: – LL_GPIO_PULL_NO – LL_GPIO_PULL_UP – LL_GPIO_PULL_DOWN
Notes	• Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• PUPDR PUPDy LL_GPIO_GetPinPull

LL_GPIO_SetAFPin_0_7

Function name	**__STATIC_INLINE void LL_GPIO_SetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)**
Function description	Configure gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 • **Alternate:** This parameter can be one of the following values: – LL_GPIO_AF_0 – LL_GPIO_AF_1 – LL_GPIO_AF_2 – LL_GPIO_AF_3

- LL_GPIO_AF_4
- LL_GPIO_AF_5
- LL_GPIO_AF_6
- LL_GPIO_AF_7
- LL_GPIO_AF_8
- LL_GPIO_AF_9
- LL_GPIO_AF_10
- LL_GPIO_AF_11
- LL_GPIO_AF_12
- LL_GPIO_AF_13
- LL_GPIO_AF_14
- LL_GPIO_AF_15

Return values	• **None:**
Notes	• Possible values are from AF0 to AF15 depending on target. • Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• AFRL AFSELy LL_GPIO_SetAFPin_0_7

LL_GPIO_GetAFPin_0_7

Function name	**__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin)**
Function description	Return gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7
Return values	• **Returned:** value can be one of the following values: – LL_GPIO_AF_0 – LL_GPIO_AF_1 – LL_GPIO_AF_2 – LL_GPIO_AF_3 – LL_GPIO_AF_4 – LL_GPIO_AF_5 – LL_GPIO_AF_6 – LL_GPIO_AF_7 – LL_GPIO_AF_8 – LL_GPIO_AF_9 – LL_GPIO_AF_10 – LL_GPIO_AF_11 – LL_GPIO_AF_12 – LL_GPIO_AF_13

- – LL_GPIO_AF_14
- – LL_GPIO_AF_15

Reference Manual to LL API cross reference:	• AFRL AFSELy LL_GPIO_GetAFPin_0_7

LL_GPIO_SetAFPin_8_15

Function name	**__STATIC_INLINE void LL_GPIO_SetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)**
Function description	Configure gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 • **Alternate:** This parameter can be one of the following values: – LL_GPIO_AF_0 – LL_GPIO_AF_1 – LL_GPIO_AF_2 – LL_GPIO_AF_3 – LL_GPIO_AF_4 – LL_GPIO_AF_5 – LL_GPIO_AF_6 – LL_GPIO_AF_7 – LL_GPIO_AF_8 – LL_GPIO_AF_9 – LL_GPIO_AF_10 – LL_GPIO_AF_11 – LL_GPIO_AF_12 – LL_GPIO_AF_13 – LL_GPIO_AF_14 – LL_GPIO_AF_15
Return values	• **None:**
Notes	• Possible values are from AF0 to AF15 depending on target. • Warning: only one pin can be passed as parameter.
Reference Manual to LL API cross reference:	• AFRH AFSELy LL_GPIO_SetAFPin_8_15

LL_GPIO_GetAFPin_8_15

Function name	**__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_8_15**

	(GPIO_TypeDef * GPIOx, uint32_t Pin)
Function description	Return gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **Pin:** This parameter can be one of the following values: – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15
Return values	• **Returned:** value can be one of the following values: – LL_GPIO_AF_0 – LL_GPIO_AF_1 – LL_GPIO_AF_2 – LL_GPIO_AF_3 – LL_GPIO_AF_4 – LL_GPIO_AF_5 – LL_GPIO_AF_6 – LL_GPIO_AF_7 – LL_GPIO_AF_8 – LL_GPIO_AF_9 – LL_GPIO_AF_10 – LL_GPIO_AF_11 – LL_GPIO_AF_12 – LL_GPIO_AF_13 – LL_GPIO_AF_14 – LL_GPIO_AF_15
Notes	• Possible values are from AF0 to AF15 depending on target.
Reference Manual to LL API cross reference:	• AFRH AFSELy LL_GPIO_GetAFPin_8_15

LL_GPIO_LockPin

Function name	**__STATIC_INLINE void LL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Lock configuration of several pins for a dedicated port.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5

- LL_GPIO_PIN_6
- LL_GPIO_PIN_7
- LL_GPIO_PIN_8
- LL_GPIO_PIN_9
- LL_GPIO_PIN_10
- LL_GPIO_PIN_11
- LL_GPIO_PIN_12
- LL_GPIO_PIN_13
- LL_GPIO_PIN_14
- LL_GPIO_PIN_15
- LL_GPIO_PIN_ALL

Return values	• **None:**
Notes	• When the lock sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset. • Each lock bit freezes a specific configuration register (control and alternate function registers).
Reference Manual to LL API cross reference:	• LCKR LCKK LL_GPIO_LockPin

LL_GPIO_IsPinLocked

Function name	**__STATIC_INLINE uint32_t LL_GPIO_IsPinLocked (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Return 1 if all pins passed as parameter, of a dedicated port, are locked.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• LCKR LCKy LL_GPIO_IsPinLocked

reference:

LL_GPIO_IsAnyPinLocked

Function name	**__STATIC_INLINE uint32_t LL_GPIO_IsAnyPinLocked (GPIO_TypeDef * GPIOx)**
Function description	Return 1 if one of the pin of a dedicated port is locked.
Parameters	• **GPIOx:** GPIO Port
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• LCKR LCKK LL_GPIO_IsAnyPinLocked

LL_GPIO_ReadInputPort

Function name	**__STATIC_INLINE uint32_t LL_GPIO_ReadInputPort (GPIO_TypeDef * GPIOx)**
Function description	Return full input data register value for a dedicated port.
Parameters	• **GPIOx:** GPIO Port
Return values	• **Input:** data register value of port
Reference Manual to LL API cross reference:	• IDR IDy LL_GPIO_ReadInputPort

LL_GPIO_IsInputPinSet

Function name	**__STATIC_INLINE uint32_t LL_GPIO_IsInputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Return if input data level for several pins of dedicated port is high or low.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14

– LL_GPIO_PIN_15
– LL_GPIO_PIN_ALL

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IDR IDy LL_GPIO_IsInputPinSet

LL_GPIO_WriteOutputPort

Function name	**__STATIC_INLINE void LL_GPIO_WriteOutputPort (GPIO_TypeDef * GPIOx, uint32_t PortValue)**
Function description	Write output data register for the port.
Parameters	• **GPIOx:** GPIO Port • **PortValue:** Level value for each pin of the port
Return values	• **None:**
Reference Manual to LL API cross reference:	• ODR ODy LL_GPIO_WriteOutputPort

LL_GPIO_ReadOutputPort

Function name	**__STATIC_INLINE uint32_t LL_GPIO_ReadOutputPort (GPIO_TypeDef * GPIOx)**
Function description	Return full output data register value for a dedicated port.
Parameters	• **GPIOx:** GPIO Port
Return values	• **Output:** data register value of port
Reference Manual to LL API cross reference:	• ODR ODy LL_GPIO_ReadOutputPort

LL_GPIO_IsOutputPinSet

Function name	**__STATIC_INLINE uint32_t LL_GPIO_IsOutputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Return if input data level for several pins of dedicated port is high or low.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8

- LL_GPIO_PIN_9
- LL_GPIO_PIN_10
- LL_GPIO_PIN_11
- LL_GPIO_PIN_12
- LL_GPIO_PIN_13
- LL_GPIO_PIN_14
- LL_GPIO_PIN_15
- LL_GPIO_PIN_ALL

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ODR ODy LL_GPIO_IsOutputPinSet

LL_GPIO_SetOutputPin

Function name	**__STATIC_INLINE void LL_GPIO_SetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Set several pins to high level on dedicated gpio port.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL
Return values	• **None:**
Reference Manual to LL API cross reference:	• BSRR BSy LL_GPIO_SetOutputPin

LL_GPIO_ResetOutputPin

Function name	**__STATIC_INLINE void LL_GPIO_ResetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Set several pins to low level on dedicated gpio port.
Parameters	• **GPIOx:** GPIO Port

- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values
- **None:**

Reference Manual to LL API cross reference:
- BRR BRy LL_GPIO_ResetOutputPin

LL_GPIO_TogglePin

Function name
__STATIC_INLINE void LL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description
Toggle data value for several pin of dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values
- **None:**

Reference Manual to LL API cross reference:	• ODR ODy LL_GPIO_TogglePin

LL_GPIO_DeInit

Function name	**ErrorStatus LL_GPIO_DeInit (GPIO_TypeDef * GPIOx)**
Function description	De-initialize GPIO registers (Registers restored to their default values).
Parameters	• **GPIOx:** GPIO Port
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: GPIO registers are de-initialized – ERROR: Wrong GPIO Port

LL_GPIO_Init

Function name	**ErrorStatus LL_GPIO_Init (GPIO_TypeDef * GPIOx, LL_GPIO_InitTypeDef * GPIO_InitStruct)**
Function description	Initialize GPIO registers according to the specified parameters in GPIO_InitStruct.
Parameters	• **GPIOx:** GPIO Port • **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: GPIO registers are initialized according to GPIO_InitStruct content – ERROR: Not applicable

LL_GPIO_StructInit

Function name	**void LL_GPIO_StructInit (LL_GPIO_InitTypeDef * GPIO_InitStruct)**
Function description	Set each LL_GPIO_InitTypeDef field to default value.
Parameters	• **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

84.3 GPIO Firmware driver defines

84.3.1 GPIO

Alternate Function

LL_GPIO_AF_0	Select alternate function 0
LL_GPIO_AF_1	Select alternate function 1
LL_GPIO_AF_2	Select alternate function 2

LL_GPIO_AF_3	Select alternate function 3
LL_GPIO_AF_4	Select alternate function 4
LL_GPIO_AF_5	Select alternate function 5
LL_GPIO_AF_6	Select alternate function 6
LL_GPIO_AF_7	Select alternate function 7
LL_GPIO_AF_8	Select alternate function 8
LL_GPIO_AF_9	Select alternate function 9
LL_GPIO_AF_10	Select alternate function 10
LL_GPIO_AF_11	Select alternate function 11
LL_GPIO_AF_12	Select alternate function 12
LL_GPIO_AF_13	Select alternate function 13
LL_GPIO_AF_14	Select alternate function 14
LL_GPIO_AF_15	Select alternate function 15

Mode

LL_GPIO_MODE_INPUT	Select input mode
LL_GPIO_MODE_OUTPUT	Select output mode
LL_GPIO_MODE_ALTERNATE	Select alternate function mode
LL_GPIO_MODE_ANALOG	Select analog mode

Output Type

LL_GPIO_OUTPUT_PUSHPULL	Select push-pull as output type
LL_GPIO_OUTPUT_OPENDRAIN	Select open-drain as output type

PIN

LL_GPIO_PIN_0	Select pin 0
LL_GPIO_PIN_1	Select pin 1
LL_GPIO_PIN_2	Select pin 2
LL_GPIO_PIN_3	Select pin 3
LL_GPIO_PIN_4	Select pin 4
LL_GPIO_PIN_5	Select pin 5
LL_GPIO_PIN_6	Select pin 6
LL_GPIO_PIN_7	Select pin 7
LL_GPIO_PIN_8	Select pin 8
LL_GPIO_PIN_9	Select pin 9
LL_GPIO_PIN_10	Select pin 10
LL_GPIO_PIN_11	Select pin 11
LL_GPIO_PIN_12	Select pin 12
LL_GPIO_PIN_13	Select pin 13

LL_GPIO_PIN_14	Select pin 14
LL_GPIO_PIN_15	Select pin 15
LL_GPIO_PIN_ALL	Select all pins

Pull Up Pull Down

LL_GPIO_PULL_NO	Select I/O no pull
LL_GPIO_PULL_UP	Select I/O pull up
LL_GPIO_PULL_DOWN	Select I/O pull down

Output Speed

LL_GPIO_SPEED_FREQ_LOW	Select I/O low output speed
LL_GPIO_SPEED_FREQ_MEDIUM	Select I/O medium output speed
LL_GPIO_SPEED_FREQ_HIGH	Select I/O fast output speed
LL_GPIO_SPEED_FREQ_VERY_HIGH	Select I/O high output speed

Common Write and read registers Macros

LL_GPIO_WriteReg

Description:

- Write a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_GPIO_ReadReg

Description:

- Read a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be read

Return value:

- Register: value

GPIO Exported Constants

LL_GPIO_SPEED_LOW

LL_GPIO_SPEED_MEDIUM

LL_GPIO_SPEED_FAST

LL_GPIO_SPEED_HIGH

85 LL I2C Generic Driver

85.1 I2C Firmware driver registers structures

85.1.1 LL_I2C_InitTypeDef

Data Fields

- ***uint32_t PeripheralMode***
- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t DigitalFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t TypeAcknowledge***
- ***uint32_t OwnAddrSize***

Field Documentation

- ***uint32_t LL_I2C_InitTypeDef::PeripheralMode***
  Specifies the peripheral mode. This parameter can be a value of *I2C_LL_EC_PERIPHERAL_MODE*This feature can be modified afterwards using unitary function **LL_I2C_SetMode()**.
- ***uint32_t LL_I2C_InitTypeDef::Timing***
  Specifies the SDA setup, hold time and the SCL high, low period values. This parameter must be set by referring to the STM32CubeMX Tool and the helper macro **__LL_I2C_CONVERT_TIMINGS()**This feature can be modified afterwards using unitary function **LL_I2C_SetTiming()**.
- ***uint32_t LL_I2C_InitTypeDef::AnalogFilter***
  Enables or disables analog noise filter. This parameter can be a value of *I2C_LL_EC_ANALOGFILTER_SELECTION*This feature can be modified afterwards using unitary functions **LL_I2C_EnableAnalogFilter()** or **LL_I2C_DisableAnalogFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::DigitalFilter***
  Configures the digital noise filter. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0x0FThis feature can be modified afterwards using unitary function **LL_I2C_SetDigitalFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddress1***
  Specifies the device own address 1. This parameter must be a value between Min_Data = 0x00 and Max_Data = 0x3FFThis feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.
- ***uint32_t LL_I2C_InitTypeDef::TypeAcknowledge***
  Specifies the ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte. This parameter can be a value of *I2C_LL_EC_I2C_ACKNOWLEDGE*This feature can be modified afterwards using unitary function **LL_I2C_AcknowledgeNextData()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddrSize***
  Specifies the device own address 1 size (7-bit or 10-bit). This parameter can be a value of *I2C_LL_EC_OWNADDRESS1*This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.

85.2 I2C Firmware driver API description

85.2.1 Detailed description of functions

LL_I2C_Enable

Function name	**__STATIC_INLINE void LL_I2C_Enable (I2C_TypeDef * I2Cx)**
Function description	Enable I2C peripheral (PE = 1).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 PE LL_I2C_Enable

LL_I2C_Disable

Function name	**__STATIC_INLINE void LL_I2C_Disable (I2C_TypeDef * I2Cx)**
Function description	Disable I2C peripheral (PE = 0).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• When PE = 0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least 3 APB clock cycles.
Reference Manual to LL API cross reference:	• CR1 PE LL_I2C_Disable

LL_I2C_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabled (I2C_TypeDef * I2Cx)**
Function description	Check if the I2C peripheral is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 PE LL_I2C_IsEnabled

LL_I2C_ConfigFilters

Function name	**__STATIC_INLINE void LL_I2C_ConfigFilters (I2C_TypeDef * I2Cx, uint32_t AnalogFilter, uint32_t DigitalFilter)**
Function description	Configure Noise Filters (Analog and Digital).
Parameters	• **I2Cx:** I2C Instance. • **AnalogFilter:** This parameter can be one of the following

values:
- – LL_I2C_ANALOGFILTER_ENABLE
- – LL_I2C_ANALOGFILTER_DISABLE
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values	• **None:**
Notes	• If the analog filter is also enabled, the digital filter is added to analog filter. The filters can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 ANFOFF LL_I2C_ConfigFilters • CR1 DNF LL_I2C_ConfigFilters

LL_I2C_SetDigitalFilter

Function name	**__STATIC_INLINE void LL_I2C_SetDigitalFilter (I2C_TypeDef * I2Cx, uint32_t DigitalFilter)**
Function description	Configure Digital Noise Filter.
Parameters	• **I2Cx:** I2C Instance. • **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.
Return values	• **None:**
Notes	• If the analog filter is also enabled, the digital filter is added to analog filter. This filter can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 DNF LL_I2C_SetDigitalFilter

LL_I2C_GetDigitalFilter

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetDigitalFilter (I2C_TypeDef * I2Cx)**
Function description	Get the current Digital Noise Filter configuration.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xF
Reference Manual to LL API cross reference:	• CR1 DNF LL_I2C_GetDigitalFilter

LL_I2C_EnableAnalogFilter

Function name	**__STATIC_INLINE void LL_I2C_EnableAnalogFilter (I2C_TypeDef * I2Cx)**
Function description	Enable Analog Noise Filter.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• This filter can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 ANFOFF LL_I2C_EnableAnalogFilter

LL_I2C_DisableAnalogFilter

Function name	**__STATIC_INLINE void LL_I2C_DisableAnalogFilter (I2C_TypeDef * I2Cx)**
Function description	Disable Analog Noise Filter.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• This filter can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 ANFOFF LL_I2C_DisableAnalogFilter

LL_I2C_IsEnabledAnalogFilter

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledAnalogFilter (I2C_TypeDef * I2Cx)**
Function description	Check if Analog Noise Filter is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 ANFOFF LL_I2C_IsEnabledAnalogFilter

LL_I2C_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_I2C_EnableDMAReq_TX (I2C_TypeDef * I2Cx)**
Function description	Enable DMA transmission requests.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to	• CR1 TXDMAEN LL_I2C_EnableDMAReq_TX

LL API cross reference:	

LL_I2C_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_I2C_DisableDMAReq_TX (I2C_TypeDef * I2Cx)**
Function description	Disable DMA transmission requests.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TXDMAEN LL_I2C_DisableDMAReq_TX

LL_I2C_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_TX (I2C_TypeDef * I2Cx)**
Function description	Check if DMA transmission requests are enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TXDMAEN LL_I2C_IsEnabledDMAReq_TX

LL_I2C_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_I2C_EnableDMAReq_RX (I2C_TypeDef * I2Cx)**
Function description	Enable DMA reception requests.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RXDMAEN LL_I2C_EnableDMAReq_RX

LL_I2C_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_I2C_DisableDMAReq_RX (I2C_TypeDef * I2Cx)**
Function description	Disable DMA reception requests.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross	• CR1 RXDMAEN LL_I2C_DisableDMAReq_RX

reference:

LL_I2C_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_RX (I2C_TypeDef * I2Cx)**
Function description	Check if DMA reception requests are enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RXDMAEN LL_I2C_IsEnabledDMAReq_RX

LL_I2C_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr (I2C_TypeDef * I2Cx, uint32_t Direction)**
Function description	Get the data register address used for DMA transfer.
Parameters	• **I2Cx:** I2C Instance • **Direction:** This parameter can be one of the following values: – LL_I2C_DMA_REG_DATA_TRANSMIT – LL_I2C_DMA_REG_DATA_RECEIVE
Return values	• **Address:** of data register
Reference Manual to LL API cross reference:	• TXDR TXDATA LL_I2C_DMA_GetRegAddr • RXDR RXDATA LL_I2C_DMA_GetRegAddr

LL_I2C_EnableClockStretching

Function name	**__STATIC_INLINE void LL_I2C_EnableClockStretching (I2C_TypeDef * I2Cx)**
Function description	Enable Clock stretching.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 NOSTRETCH LL_I2C_EnableClockStretching

LL_I2C_DisableClockStretching

Function name	**__STATIC_INLINE void LL_I2C_DisableClockStretching (I2C_TypeDef * I2Cx)**
Function description	Disable Clock stretching.

Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 NOSTRETCH LL_I2C_DisableClockStretching

LL_I2C_IsEnabledClockStretching

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledClockStretching (I2C_TypeDef * I2Cx)**
Function description	Check if Clock stretching is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 NOSTRETCH LL_I2C_IsEnabledClockStretching

LL_I2C_EnableSlaveByteControl

Function name	**__STATIC_INLINE void LL_I2C_EnableSlaveByteControl (I2C_TypeDef * I2Cx)**
Function description	Enable hardware byte control in slave mode.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 SBC LL_I2C_EnableSlaveByteControl

LL_I2C_DisableSlaveByteControl

Function name	**__STATIC_INLINE void LL_I2C_DisableSlaveByteControl (I2C_TypeDef * I2Cx)**
Function description	Disable hardware byte control in slave mode.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 SBC LL_I2C_DisableSlaveByteControl

LL_I2C_IsEnabledSlaveByteControl

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSlaveByteControl (I2C_TypeDef * I2Cx)**

Function description	Check if hardware byte control in slave mode is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 SBC LL_I2C_IsEnabledSlaveByteControl

LL_I2C_EnableWakeUpFromStop

Function name	**__STATIC_INLINE void LL_I2C_EnableWakeUpFromStop (I2C_TypeDef * I2Cx)**
Function description	Enable Wakeup from STOP.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance. • This bit can only be programmed when Digital Filter is disabled.
Reference Manual to LL API cross reference:	• CR1 WUPEN LL_I2C_EnableWakeUpFromStop

LL_I2C_DisableWakeUpFromStop

Function name	**__STATIC_INLINE void LL_I2C_DisableWakeUpFromStop (I2C_TypeDef * I2Cx)**
Function description	Disable Wakeup from STOP.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 WUPEN LL_I2C_DisableWakeUpFromStop

LL_I2C_IsEnabledWakeUpFromStop

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledWakeUpFromStop (I2C_TypeDef * I2Cx)**
Function description	Check if Wakeup from STOP is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).

Notes	• Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 WUPEN LL_I2C_IsEnabledWakeUpFromStop

LL_I2C_EnableGeneralCall

Function name	**__STATIC_INLINE void LL_I2C_EnableGeneralCall (I2C_TypeDef * I2Cx)**
Function description	Enable General Call.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• When enabled the Address 0x00 is ACKed.
Reference Manual to LL API cross reference:	• CR1 GCEN LL_I2C_EnableGeneralCall

LL_I2C_DisableGeneralCall

Function name	**__STATIC_INLINE void LL_I2C_DisableGeneralCall (I2C_TypeDef * I2Cx)**
Function description	Disable General Call.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• When disabled the Address 0x00 is NACKed.
Reference Manual to LL API cross reference:	• CR1 GCEN LL_I2C_DisableGeneralCall

LL_I2C_IsEnabledGeneralCall

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledGeneralCall (I2C_TypeDef * I2Cx)**
Function description	Check if General Call is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 GCEN LL_I2C_IsEnabledGeneralCall

LL_I2C_SetMasterAddressingMode

Function name	**__STATIC_INLINE void LL_I2C_SetMasterAddressingMode (I2C_TypeDef * I2Cx, uint32_t AddressingMode)**

Function description	Configure the Master to operate in 7-bit or 10-bit addressing mode.
Parameters	• **I2Cx:** I2C Instance. • **AddressingMode:** This parameter can be one of the following values: – LL_I2C_ADDRESSING_MODE_7BIT – LL_I2C_ADDRESSING_MODE_10BIT
Return values	• **None:**
Notes	• Changing this bit is not allowed, when the START bit is set.
Reference Manual to LL API cross reference:	• CR2 ADD10 LL_I2C_SetMasterAddressingMode

LL_I2C_GetMasterAddressingMode

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetMasterAddressingMode (I2C_TypeDef * I2Cx)**
Function description	Get the Master addressing mode.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_ADDRESSING_MODE_7BIT – LL_I2C_ADDRESSING_MODE_10BIT
Reference Manual to LL API cross reference:	• CR2 ADD10 LL_I2C_GetMasterAddressingMode

LL_I2C_SetOwnAddress1

Function name	**__STATIC_INLINE void LL_I2C_SetOwnAddress1 (I2C_TypeDef * I2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)**
Function description	Set the Own Address1.
Parameters	• **I2Cx:** I2C Instance. • **OwnAddress1:** This parameter must be a value between Min_Data=0 and Max_Data=0x3FF. • **OwnAddrSize:** This parameter can be one of the following values: – LL_I2C_OWNADDRESS1_7BIT – LL_I2C_OWNADDRESS1_10BIT
Return values	• **None:**
Reference Manual to LL API cross reference:	• OAR1 OA1 LL_I2C_SetOwnAddress1 • OAR1 OA1MODE LL_I2C_SetOwnAddress1

LL_I2C_EnableOwnAddress1

Function name	**__STATIC_INLINE void LL_I2C_EnableOwnAddress1 (I2C_TypeDef * I2Cx)**

Function description	Enable acknowledge on Own Address1 match address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• OAR1 OA1EN LL_I2C_EnableOwnAddress1

LL_I2C_DisableOwnAddress1

Function name	**__STATIC_INLINE void LL_I2C_DisableOwnAddress1 (I2C_TypeDef * I2Cx)**
Function description	Disable acknowledge on Own Address1 match address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• OAR1 OA1EN LL_I2C_DisableOwnAddress1

LL_I2C_IsEnabledOwnAddress1

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress1 (I2C_TypeDef * I2Cx)**
Function description	Check if Own Address1 acknowledge is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• OAR1 OA1EN LL_I2C_IsEnabledOwnAddress1

LL_I2C_SetOwnAddress2

Function name	**__STATIC_INLINE void LL_I2C_SetOwnAddress2 (I2C_TypeDef * I2Cx, uint32_t OwnAddress2, uint32_t OwnAddrMask)**
Function description	Set the 7bits Own Address2.
Parameters	• **I2Cx:** I2C Instance. • **OwnAddress2:** Value between Min_Data=0 and Max_Data=0x7F. • **OwnAddrMask:** This parameter can be one of the following values: – LL_I2C_OWNADDRESS2_NOMASK – LL_I2C_OWNADDRESS2_MASK01 – LL_I2C_OWNADDRESS2_MASK02 – LL_I2C_OWNADDRESS2_MASK03 – LL_I2C_OWNADDRESS2_MASK04 – LL_I2C_OWNADDRESS2_MASK05 – LL_I2C_OWNADDRESS2_MASK06

	– LL_I2C_OWNADDRESS2_MASK07
Return values	• **None:**
Notes	• This action has no effect if own address2 is enabled.
Reference Manual to LL API cross reference:	• OAR2 OA2 LL_I2C_SetOwnAddress2 • OAR2 OA2MSK LL_I2C_SetOwnAddress2

LL_I2C_EnableOwnAddress2

Function name	**__STATIC_INLINE void LL_I2C_EnableOwnAddress2 (I2C_TypeDef * I2Cx)**
Function description	Enable acknowledge on Own Address2 match address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• OAR2 OA2EN LL_I2C_EnableOwnAddress2

LL_I2C_DisableOwnAddress2

Function name	**__STATIC_INLINE void LL_I2C_DisableOwnAddress2 (I2C_TypeDef * I2Cx)**
Function description	Disable acknowledge on Own Address2 match address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• OAR2 OA2EN LL_I2C_DisableOwnAddress2

LL_I2C_IsEnabledOwnAddress2

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress2 (I2C_TypeDef * I2Cx)**
Function description	Check if Own Address1 acknowledge is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• OAR2 OA2EN LL_I2C_IsEnabledOwnAddress2

LL_I2C_SetTiming

Function name	**__STATIC_INLINE void LL_I2C_SetTiming (I2C_TypeDef * I2Cx, uint32_t Timing)**
Function description	Configure the SDA setup, hold time and the SCL high, low period.

Parameters	• **I2Cx:** I2C Instance. • **Timing:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.
Return values	• **None:**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0). • This parameter is computed with the STM32CubeMX Tool.
Reference Manual to LL API cross reference:	• TIMINGR TIMINGR LL_I2C_SetTiming

LL_I2C_GetTimingPrescaler

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetTimingPrescaler (I2C_TypeDef * I2Cx)**
Function description	Get the Timing Prescaler setting.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xF
Reference Manual to LL API cross reference:	• TIMINGR PRESC LL_I2C_GetTimingPrescaler

LL_I2C_GetClockLowPeriod

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetClockLowPeriod (I2C_TypeDef * I2Cx)**
Function description	Get the SCL low period setting.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• TIMINGR SCLL LL_I2C_GetClockLowPeriod

LL_I2C_GetClockHighPeriod

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetClockHighPeriod (I2C_TypeDef * I2Cx)**
Function description	Get the SCL high period setting.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• TIMINGR SCLH LL_I2C_GetClockHighPeriod

LL_I2C_GetDataHoldTime

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetDataHoldTime (I2C_TypeDef * I2Cx)**
Function description	Get the SDA hold time.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xF
Reference Manual to LL API cross reference:	• TIMINGR SDADEL LL_I2C_GetDataHoldTime

LL_I2C_GetDataSetupTime

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetDataSetupTime (I2C_TypeDef * I2Cx)**
Function description	Get the SDA setup time.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xF
Reference Manual to LL API cross reference:	• TIMINGR SCLDEL LL_I2C_GetDataSetupTime

LL_I2C_SetMode

Function name	**__STATIC_INLINE void LL_I2C_SetMode (I2C_TypeDef * I2Cx, uint32_t PeripheralMode)**
Function description	Configure peripheral mode.
Parameters	• **I2Cx:** I2C Instance. • **PeripheralMode:** This parameter can be one of the following values: – LL_I2C_MODE_I2C – LL_I2C_MODE_SMBUS_HOST – LL_I2C_MODE_SMBUS_DEVICE – LL_I2C_MODE_SMBUS_DEVICE_ARP
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 SMBHEN LL_I2C_SetMode • CR1 SMBDEN LL_I2C_SetMode

LL_I2C_GetMode

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetMode (I2C_TypeDef * I2Cx)**
Function description	Get peripheral mode.

Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_MODE_I2C – LL_I2C_MODE_SMBUS_HOST – LL_I2C_MODE_SMBUS_DEVICE – LL_I2C_MODE_SMBUS_DEVICE_ARP
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 SMBHEN LL_I2C_GetMode • CR1 SMBDEN LL_I2C_GetMode

LL_I2C_EnableSMBusAlert

Function name	**__STATIC_INLINE void LL_I2C_EnableSMBusAlert (I2C_TypeDef * I2Cx)**
Function description	Enable SMBus alert (Host or Device mode)
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • SMBus Device mode: SMBus Alert pin is drived low and Alert Response Address Header acknowledge is enabled. SMBus Host mode:SMBus Alert pin management is supported.
Reference Manual to LL API cross reference:	• CR1 ALERTEN LL_I2C_EnableSMBusAlert

LL_I2C_DisableSMBusAlert

Function name	**__STATIC_INLINE void LL_I2C_DisableSMBusAlert (I2C_TypeDef * I2Cx)**
Function description	Disable SMBus alert (Host or Device mode)
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • SMBus Device mode: SMBus Alert pin is not drived (can be used as a standard GPIO) and Alert Response Address Header acknowledge is disabled. SMBus Host mode:SMBus Alert pin management is not supported.
Reference Manual to LL API cross reference:	• CR1 ALERTEN LL_I2C_DisableSMBusAlert

LL_I2C_IsEnabledSMBusAlert

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusAlert (I2C_TypeDef * I2Cx)**
Function description	Check if SMBus alert (Host or Device mode) is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 ALERTEN LL_I2C_IsEnabledSMBusAlert

LL_I2C_EnableSMBusPEC

Function name	**__STATIC_INLINE void LL_I2C_EnableSMBusPEC (I2C_TypeDef * I2Cx)**
Function description	Enable SMBus Packet Error Calculation (PEC).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 PECEN LL_I2C_EnableSMBusPEC

LL_I2C_DisableSMBusPEC

Function name	**__STATIC_INLINE void LL_I2C_DisableSMBusPEC (I2C_TypeDef * I2Cx)**
Function description	Disable SMBus Packet Error Calculation (PEC).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 PECEN LL_I2C_DisableSMBusPEC

LL_I2C_IsEnabledSMBusPEC

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPEC**

	(I2C_TypeDef * I2Cx)
Function description	Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 PECEN LL_I2C_IsEnabledSMBusPEC

LL_I2C_ConfigSMBusTimeout

Function name	**__STATIC_INLINE void LL_I2C_ConfigSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t TimeoutA, uint32_t TimeoutAMode, uint32_t TimeoutB)**
Function description	Configure the SMBus Clock Timeout.
Parameters	• **I2Cx:** I2C Instance. • **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF. • **TimeoutAMode:** This parameter can be one of the following values: – LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW – LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH • **TimeoutB:**
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • This configuration can only be programmed when associated Timeout is disabled (TimeoutA and/orTimeoutB).
Reference Manual to LL API cross reference:	• TIMEOUTR TIMEOUTA LL_I2C_ConfigSMBusTimeout • TIMEOUTR TIDLE LL_I2C_ConfigSMBusTimeout • TIMEOUTR TIMEOUTB LL_I2C_ConfigSMBusTimeout

LL_I2C_SetSMBusTimeoutA

Function name	**__STATIC_INLINE void LL_I2C_SetSMBusTimeoutA (I2C_TypeDef * I2Cx, uint32_t TimeoutA)**
Function description	Configure the SMBus Clock TimeoutA (SCL low timeout or SCL and SDA high timeout depends on TimeoutA mode).
Parameters	• **I2Cx:** I2C Instance. • **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.
Return values	• **None:**

Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • These bits can only be programmed when TimeoutA is disabled.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMEOUTA LL_I2C_SetSMBusTimeoutA

LL_I2C_GetSMBusTimeoutA

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutA (I2C_TypeDef * I2Cx)**
Function description	Get the SMBus Clock TimeoutA setting.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0 and Max_Data=0xFFF
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMEOUTA LL_I2C_GetSMBusTimeoutA

LL_I2C_SetSMBusTimeoutAMode

Function name	**__STATIC_INLINE void LL_I2C_SetSMBusTimeoutAMode (I2C_TypeDef * I2Cx, uint32_t TimeoutAMode)**
Function description	Set the SMBus Clock TimeoutA mode.
Parameters	• **I2Cx:** I2C Instance. • **TimeoutAMode:** This parameter can be one of the following values: – LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW – LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • This bit can only be programmed when TimeoutA is disabled.
Reference Manual to LL API cross reference:	• TIMEOUTR TIDLE LL_I2C_SetSMBusTimeoutAMode

LL_I2C_GetSMBusTimeoutAMode

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutAMode (I2C_TypeDef * I2Cx)**
Function	Get the SMBus Clock TimeoutA mode.

description	
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW – LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIDLE LL_I2C_GetSMBusTimeoutAMode

LL_I2C_SetSMBusTimeoutB

Function name	**__STATIC_INLINE void LL_I2C_SetSMBusTimeoutB (I2C_TypeDef * I2Cx, uint32_t TimeoutB)**
Function description	Configure the SMBus Extended Cumulative Clock TimeoutB (Master or Slave mode).
Parameters	• **I2Cx:** I2C Instance. • **TimeoutB:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • These bits can only be programmed when TimeoutB is disabled.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMEOUTB LL_I2C_SetSMBusTimeoutB

LL_I2C_GetSMBusTimeoutB

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutB (I2C_TypeDef * I2Cx)**
Function description	Get the SMBus Extented Cumulative Clock TimeoutB setting.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0 and Max_Data=0xFFF
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMEOUTB LL_I2C_GetSMBusTimeoutB

LL_I2C_EnableSMBusTimeout

Function name	**__STATIC_INLINE void LL_I2C_EnableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)**
Function description	Enable the SMBus Clock Timeout.
Parameters	• **I2Cx:** I2C Instance. • **ClockTimeout:** This parameter can be one of the following values: – LL_I2C_SMBUS_TIMEOUTA – LL_I2C_SMBUS_TIMEOUTB – LL_I2C_SMBUS_ALL_TIMEOUT
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMOUTEN LL_I2C_EnableSMBusTimeout • TIMEOUTR TEXTEN LL_I2C_EnableSMBusTimeout

LL_I2C_DisableSMBusTimeout

Function name	**__STATIC_INLINE void LL_I2C_DisableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)**
Function description	Disable the SMBus Clock Timeout.
Parameters	• **I2Cx:** I2C Instance. • **ClockTimeout:** This parameter can be one of the following values: – LL_I2C_SMBUS_TIMEOUTA – LL_I2C_SMBUS_TIMEOUTB – LL_I2C_SMBUS_ALL_TIMEOUT
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMOUTEN LL_I2C_DisableSMBusTimeout • TIMEOUTR TEXTEN LL_I2C_DisableSMBusTimeout

LL_I2C_IsEnabledSMBusTimeout

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)**
Function description	Check if the SMBus Clock Timeout is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance. • **ClockTimeout:** This parameter can be one of the following values: – LL_I2C_SMBUS_TIMEOUTA – LL_I2C_SMBUS_TIMEOUTB

	– LL_I2C_SMBUS_ALL_TIMEOUT
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• TIMEOUTR TIMOUTEN LL_I2C_IsEnabledSMBusTimeout • TIMEOUTR TEXTEN LL_I2C_IsEnabledSMBusTimeout

LL_I2C_EnableIT_TX

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_TX (I2C_TypeDef * I2Cx)**
Function description	Enable TXIS interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TXIE LL_I2C_EnableIT_TX

LL_I2C_DisableIT_TX

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_TX (I2C_TypeDef * I2Cx)**
Function description	Disable TXIS interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TXIE LL_I2C_DisableIT_TX

LL_I2C_IsEnabledIT_TX

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TX (I2C_TypeDef * I2Cx)**
Function description	Check if the TXIS Interrupt is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TXIE LL_I2C_IsEnabledIT_TX

LL_I2C_EnableIT_RX

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_RX (I2C_TypeDef * I2Cx)**

Function description	Enable RXNE interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RXIE LL_I2C_EnableIT_RX

LL_I2C_DisableIT_RX

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_RX (I2C_TypeDef * I2Cx)**
Function description	Disable RXNE interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RXIE LL_I2C_DisableIT_RX

LL_I2C_IsEnabledIT_RX

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_RX (I2C_TypeDef * I2Cx)**
Function description	Check if the RXNE Interrupt is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RXIE LL_I2C_IsEnabledIT_RX

LL_I2C_EnableIT_ADDR

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_ADDR (I2C_TypeDef * I2Cx)**
Function description	Enable Address match interrupt (slave mode only).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 ADDRIE LL_I2C_EnableIT_ADDR

LL_I2C_DisableIT_ADDR

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_ADDR (I2C_TypeDef * I2Cx)**
Function description	Disable Address match interrupt (slave mode only).

Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 ADDRIE LL_I2C_DisableIT_ADDR

LL_I2C_IsEnabledIT_ADDR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ADDR (I2C_TypeDef * I2Cx)**
Function description	Check if Address match interrupt is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 ADDRIE LL_I2C_IsEnabledIT_ADDR

LL_I2C_EnableIT_NACK

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_NACK (I2C_TypeDef * I2Cx)**
Function description	Enable Not acknowledge received interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 NACKIE LL_I2C_EnableIT_NACK

LL_I2C_DisableIT_NACK

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_NACK (I2C_TypeDef * I2Cx)**
Function description	Disable Not acknowledge received interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 NACKIE LL_I2C_DisableIT_NACK

LL_I2C_IsEnabledIT_NACK

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_NACK (I2C_TypeDef * I2Cx)**
Function description	Check if Not acknowledge received interrupt is enabled or disabled.

Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 NACKIE LL_I2C_IsEnabledIT_NACK

LL_I2C_EnableIT_STOP

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_STOP (I2C_TypeDef * I2Cx)**
Function description	Enable STOP detection interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 STOPIE LL_I2C_EnableIT_STOP

LL_I2C_DisableIT_STOP

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_STOP (I2C_TypeDef * I2Cx)**
Function description	Disable STOP detection interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 STOPIE LL_I2C_DisableIT_STOP

LL_I2C_IsEnabledIT_STOP

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_STOP (I2C_TypeDef * I2Cx)**
Function description	Check if STOP detection interrupt is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 STOPIE LL_I2C_IsEnabledIT_STOP

LL_I2C_EnableIT_TC

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_TC (I2C_TypeDef * I2Cx)**
Function description	Enable Transfer Complete interrupt.
Parameters	• **I2Cx:** I2C Instance.

Return values	• **None:**
Notes	• Any of these events will generate interrupt: Transfer Complete (TC) Transfer Complete Reload (TCR)
Reference Manual to LL API cross reference:	• CR1 TCIE LL_I2C_EnableIT_TC

LL_I2C_DisableIT_TC

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_TC (I2C_TypeDef * I2Cx)**
Function description	Disable Transfer Complete interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Any of these events will generate interrupt: Transfer Complete (TC) Transfer Complete Reload (TCR)
Reference Manual to LL API cross reference:	• CR1 TCIE LL_I2C_DisableIT_TC

LL_I2C_IsEnabledIT_TC

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TC (I2C_TypeDef * I2Cx)**
Function description	Check if Transfer Complete interrupt is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TCIE LL_I2C_IsEnabledIT_TC

LL_I2C_EnableIT_ERR

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_ERR (I2C_TypeDef * I2Cx)**
Function description	Enable Error interrupts.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • Any of these errors will generate interrupt: Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:	• CR1 ERRIE LL_I2C_EnableIT_ERR

LL_I2C_DisableIT_ERR

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_ERR (I2C_TypeDef * I2Cx)**
Function description	Disable Error interrupts.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • Any of these errors will generate interrupt: Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)
Reference Manual to LL API cross reference:	• CR1 ERRIE LL_I2C_DisableIT_ERR

LL_I2C_IsEnabledIT_ERR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ERR (I2C_TypeDef * I2Cx)**
Function description	Check if Error interrupts are enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 ERRIE LL_I2C_IsEnabledIT_ERR

LL_I2C_IsActiveFlag_TXE

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXE (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Transmit data register empty flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.
Reference Manual to LL API cross reference:	• ISR TXE LL_I2C_IsActiveFlag_TXE

LL_I2C_IsActiveFlag_TXIS

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXIS (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Transmit interrupt flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.
Reference Manual to LL API cross reference:	• ISR TXIS LL_I2C_IsActiveFlag_TXIS

LL_I2C_IsActiveFlag_RXNE

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_RXNE (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Receive data register not empty flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: When Receive data register is read. SET: When the received data is copied in Receive data register.
Reference Manual to LL API cross reference:	• ISR RXNE LL_I2C_IsActiveFlag_RXNE

LL_I2C_IsActiveFlag_ADDR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Address matched flag (slave mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When the received slave address matched with one of the enabled slave address.
Reference Manual to LL API cross reference:	• ISR ADDR LL_I2C_IsActiveFlag_ADDR

LL_I2C_IsActiveFlag_NACK

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_NACK (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Not Acknowledge received flag.
Parameters	• **I2Cx:** I2C Instance.

Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a NACK is received after a byte transmission.
Reference Manual to LL API cross reference:	• ISR NACKF LL_I2C_IsActiveFlag_NACK

LL_I2C_IsActiveFlag_STOP

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_STOP (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Stop detection flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a Stop condition is detected.
Reference Manual to LL API cross reference:	• ISR STOPF LL_I2C_IsActiveFlag_STOP

LL_I2C_IsActiveFlag_TC

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TC (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Transfer complete flag (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When RELOAD=0, AUTOEND=0 and NBYTES date have been transferred.
Reference Manual to LL API cross reference:	• ISR TC LL_I2C_IsActiveFlag_TC

LL_I2C_IsActiveFlag_TCR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TCR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Transfer complete flag (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When RELOAD=1 and NBYTES date have been transferred.
Reference Manual to LL API cross reference:	• ISR TCR LL_I2C_IsActiveFlag_TCR

LL_I2C_IsActiveFlag_BERR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BERR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Bus error flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a misplaced Start or Stop condition is detected.
Reference Manual to LL API cross reference:	• ISR BERR LL_I2C_IsActiveFlag_BERR

LL_I2C_IsActiveFlag_ARLO

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ARLO (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Arbitration lost flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When arbitration lost.
Reference Manual to LL API cross reference:	• ISR ARLO LL_I2C_IsActiveFlag_ARLO

LL_I2C_IsActiveFlag_OVR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_OVR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Overrun/Underrun flag (slave mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When an overrun/underrun error occurs (Clock Stretching Disabled).
Reference Manual to LL API cross reference:	• ISR OVR LL_I2C_IsActiveFlag_OVR

LL_I2C_IsActiveSMBusFlag_PECERR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_PECERR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus PEC error flag in reception.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).

Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • RESET: Clear default value. SET: When the received PEC does not match with the PEC register content.
Reference Manual to LL API cross reference:	• ISR PECERR LL_I2C_IsActiveSMBusFlag_PECERR

LL_I2C_IsActiveSMBusFlag_TIMEOUT

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus Timeout detection flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • RESET: Clear default value. SET: When a timeout or extended clock timeout occurs.
Reference Manual to LL API cross reference:	• ISR TIMEOUT LL_I2C_IsActiveSMBusFlag_TIMEOUT

LL_I2C_IsActiveSMBusFlag_ALERT

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_ALERT (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus alert flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • RESET: Clear default value. SET: When SMBus host configuration, SMBus alert enabled and a falling edge event occurs on SMBA pin.
Reference Manual to LL API cross reference:	• ISR ALERT LL_I2C_IsActiveSMBusFlag_ALERT

LL_I2C_IsActiveFlag_BUSY

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BUSY (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Bus Busy flag.

Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a Start condition is detected.
Reference Manual to LL API cross reference:	• ISR BUSY LL_I2C_IsActiveFlag_BUSY

LL_I2C_ClearFlag_ADDR

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_ADDR (I2C_TypeDef * I2Cx)**
Function description	Clear Address Matched flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR ADDRCF LL_I2C_ClearFlag_ADDR

LL_I2C_ClearFlag_NACK

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_NACK (I2C_TypeDef * I2Cx)**
Function description	Clear Not Acknowledge flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR NACKCF LL_I2C_ClearFlag_NACK

LL_I2C_ClearFlag_STOP

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_STOP (I2C_TypeDef * I2Cx)**
Function description	Clear Stop detection flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR STOPCF LL_I2C_ClearFlag_STOP

LL_I2C_ClearFlag_TXE

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_TXE (I2C_TypeDef * I2Cx)**

Function description	Clear Transmit data register empty flag (TXE).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• This bit can be clear by software in order to flush the transmit data register (TXDR).
Reference Manual to LL API cross reference:	• ISR TXE LL_I2C_ClearFlag_TXE

LL_I2C_ClearFlag_BERR

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_BERR (I2C_TypeDef * I2Cx)**
Function description	Clear Bus error flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR BERRCF LL_I2C_ClearFlag_BERR

LL_I2C_ClearFlag_ARLO

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_ARLO (I2C_TypeDef * I2Cx)**
Function description	Clear Arbitration lost flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR ARLOCF LL_I2C_ClearFlag_ARLO

LL_I2C_ClearFlag_OVR

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_OVR (I2C_TypeDef * I2Cx)**
Function description	Clear Overrun/Underrun flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR OVRCF LL_I2C_ClearFlag_OVR

LL_I2C_ClearSMBusFlag_PECERR

Function name	**__STATIC_INLINE void LL_I2C_ClearSMBusFlag_PECERR**

	(I2C_TypeDef * I2Cx)
Function description	Clear SMBus PEC error flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• ICR PECCF LL_I2C_ClearSMBusFlag_PECERR

LL_I2C_ClearSMBusFlag_TIMEOUT

Function name	**__STATIC_INLINE void LL_I2C_ClearSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)**
Function description	Clear SMBus Timeout detection flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• ICR TIMOUTCF LL_I2C_ClearSMBusFlag_TIMEOUT

LL_I2C_ClearSMBusFlag_ALERT

Function name	**__STATIC_INLINE void LL_I2C_ClearSMBusFlag_ALERT (I2C_TypeDef * I2Cx)**
Function description	Clear SMBus Alert flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• ICR ALERTCF LL_I2C_ClearSMBusFlag_ALERT

LL_I2C_EnableAutoEndMode

Function name	**__STATIC_INLINE void LL_I2C_EnableAutoEndMode (I2C_TypeDef * I2Cx)**
Function description	Enable automatic STOP condition generation (master mode).
Parameters	• **I2Cx:** I2C Instance.

Return values	• **None:**
Notes	• Automatic end mode: a STOP condition is automatically sent when NBYTES data are transferred. This bit has no effect in slave mode or when RELOAD bit is set.
Reference Manual to LL API cross reference:	• CR2 AUTOEND LL_I2C_EnableAutoEndMode

LL_I2C_DisableAutoEndMode

Function name	**__STATIC_INLINE void LL_I2C_DisableAutoEndMode (I2C_TypeDef * I2Cx)**
Function description	Disable automatic STOP condition generation (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Software end mode: TC flag is set when NBYTES data are transferre, stretching SCL low.
Reference Manual to LL API cross reference:	• CR2 AUTOEND LL_I2C_DisableAutoEndMode

LL_I2C_IsEnabledAutoEndMode

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoEndMode (I2C_TypeDef * I2Cx)**
Function description	Check if automatic STOP condition is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 AUTOEND LL_I2C_IsEnabledAutoEndMode

LL_I2C_EnableReloadMode

Function name	**__STATIC_INLINE void LL_I2C_EnableReloadMode (I2C_TypeDef * I2Cx)**
Function description	Enable reload mode (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• The transfer is not completed after the NBYTES data transfer, NBYTES will be reloaded when TCR flag is set.
Reference Manual to LL API cross reference:	• CR2 RELOAD LL_I2C_EnableReloadMode

LL_I2C_DisableReloadMode

Function name	**__STATIC_INLINE void LL_I2C_DisableReloadMode (I2C_TypeDef * I2Cx)**
Function description	Disable reload mode (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• The transfer is completed after the NBYTES data transfer(STOP or RESTART will follow).
Reference Manual to LL API cross reference:	• CR2 RELOAD LL_I2C_DisableReloadMode

LL_I2C_IsEnabledReloadMode

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledReloadMode (I2C_TypeDef * I2Cx)**
Function description	Check if reload mode is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RELOAD LL_I2C_IsEnabledReloadMode

LL_I2C_SetTransferSize

Function name	**__STATIC_INLINE void LL_I2C_SetTransferSize (I2C_TypeDef * I2Cx, uint32_t TransferSize)**
Function description	Configure the number of bytes for transfer.
Parameters	• **I2Cx:** I2C Instance. • **TransferSize:** This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
Return values	• **None:**
Notes	• Changing these bits when START bit is set is not allowed.
Reference Manual to LL API cross reference:	• CR2 NBYTES LL_I2C_SetTransferSize

LL_I2C_GetTransferSize

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetTransferSize (I2C_TypeDef * I2Cx)**
Function description	Get the number of bytes configured for transfer.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xFF

Reference Manual to LL API cross reference:	• CR2 NBYTES LL_I2C_GetTransferSize

LL_I2C_AcknowledgeNextData

Function name	**__STATIC_INLINE void LL_I2C_AcknowledgeNextData (I2C_TypeDef * I2Cx, uint32_t TypeAcknowledge)**
Function description	Prepare the generation of a ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte.
Parameters	• **I2Cx:** I2C Instance. • **TypeAcknowledge:** This parameter can be one of the following values: – LL_I2C_ACK – LL_I2C_NACK
Return values	• **None:**
Notes	• Usage in Slave mode only.
Reference Manual to LL API cross reference:	• CR2 NACK LL_I2C_AcknowledgeNextData

LL_I2C_GenerateStartCondition

Function name	**__STATIC_INLINE void LL_I2C_GenerateStartCondition (I2C_TypeDef * I2Cx)**
Function description	Generate a START or RESTART condition.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• The START bit can be set even if bus is BUSY or I2C is in slave mode. This action has no effect when RELOAD is set.
Reference Manual to LL API cross reference:	• CR2 START LL_I2C_GenerateStartCondition

LL_I2C_GenerateStopCondition

Function name	**__STATIC_INLINE void LL_I2C_GenerateStopCondition (I2C_TypeDef * I2Cx)**
Function description	Generate a STOP condition after the current byte transfer (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 STOP LL_I2C_GenerateStopCondition

LL_I2C_EnableAuto10BitRead

Function name	**__STATIC_INLINE void LL_I2C_EnableAuto10BitRead (I2C_TypeDef * I2Cx)**
Function description	Enable automatic RESTART Read request condition for 10bit address header (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• The master sends the complete 10bit slave address read sequence: Start + 2 bytes 10bit address in Write direction + Restart + first 7 bits of 10bit address in Read direction.
Reference Manual to LL API cross reference:	• CR2 HEAD10R LL_I2C_EnableAuto10BitRead

LL_I2C_DisableAuto10BitRead

Function name	**__STATIC_INLINE void LL_I2C_DisableAuto10BitRead (I2C_TypeDef * I2Cx)**
Function description	Disable automatic RESTART Read request condition for 10bit address header (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• The master only sends the first 7 bits of 10bit address in Read direction.
Reference Manual to LL API cross reference:	• CR2 HEAD10R LL_I2C_DisableAuto10BitRead

LL_I2C_IsEnabledAuto10BitRead

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledAuto10BitRead (I2C_TypeDef * I2Cx)**
Function description	Check if automatic RESTART Read request condition for 10bit address header is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 HEAD10R LL_I2C_IsEnabledAuto10BitRead

LL_I2C_SetTransferRequest

Function name	**__STATIC_INLINE void LL_I2C_SetTransferRequest (I2C_TypeDef * I2Cx, uint32_t TransferRequest)**
Function description	Configure the transfer direction (master mode).

Parameters	• **I2Cx:** I2C Instance. • **TransferRequest:** This parameter can be one of the following values: – LL_I2C_REQUEST_WRITE – LL_I2C_REQUEST_READ
Return values	• **None:**
Notes	• Changing these bits when START bit is set is not allowed.
Reference Manual to LL API cross reference:	• CR2 RD_WRN LL_I2C_SetTransferRequest

LL_I2C_GetTransferRequest

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetTransferRequest (I2C_TypeDef * I2Cx)**
Function description	Get the transfer direction requested (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_REQUEST_WRITE – LL_I2C_REQUEST_READ
Reference Manual to LL API cross reference:	• CR2 RD_WRN LL_I2C_GetTransferRequest

LL_I2C_SetSlaveAddr

Function name	**__STATIC_INLINE void LL_I2C_SetSlaveAddr (I2C_TypeDef * I2Cx, uint32_t SlaveAddr)**
Function description	Configure the slave address for transfer (master mode).
Parameters	• **I2Cx:** I2C Instance. • **SlaveAddr:** This parameter must be a value between Min_Data=0x00 and Max_Data=0x3F.
Return values	• **None:**
Notes	• Changing these bits when START bit is set is not allowed.
Reference Manual to LL API cross reference:	• CR2 SADD LL_I2C_SetSlaveAddr

LL_I2C_GetSlaveAddr

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetSlaveAddr (I2C_TypeDef * I2Cx)**
Function description	Get the slave address programmed for transfer.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:	• CR2 SADD LL_I2C_GetSlaveAddr

LL_I2C_HandleTransfer

Function name	**__STATIC_INLINE void LL_I2C_HandleTransfer (I2C_TypeDef * I2Cx, uint32_t SlaveAddr, uint32_t SlaveAddrSize, uint32_t TransferSize, uint32_t EndMode, uint32_t Request)**
Function description	Handles I2Cx communication when starting transfer or during transfer (TC or TCR flag are set).
Parameters	• **I2Cx:** I2C Instance. • **SlaveAddr:** Specifies the slave address to be programmed. • **SlaveAddrSize:** This parameter can be one of the following values: – LL_I2C_ADDRSLAVE_7BIT – LL_I2C_ADDRSLAVE_10BIT • **TransferSize:** Specifies the number of bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=255. • **EndMode:** This parameter can be one of the following values: – LL_I2C_MODE_RELOAD – LL_I2C_MODE_AUTOEND – LL_I2C_MODE_SOFTEND – LL_I2C_MODE_SMBUS_RELOAD – LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC – LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC – LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC – LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC • **Request:** This parameter can be one of the following values: – LL_I2C_GENERATE_NOSTARTSTOP – LL_I2C_GENERATE_STOP – LL_I2C_GENERATE_START_READ – LL_I2C_GENERATE_START_WRITE – LL_I2C_GENERATE_RESTART_7BIT_READ – LL_I2C_GENERATE_RESTART_7BIT_WRITE – LL_I2C_GENERATE_RESTART_10BIT_READ – LL_I2C_GENERATE_RESTART_10BIT_WRITE
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 SADD LL_I2C_HandleTransfer • CR2 ADD10 LL_I2C_HandleTransfer • CR2 RD_WRN LL_I2C_HandleTransfer • CR2 START LL_I2C_HandleTransfer • CR2 STOP LL_I2C_HandleTransfer • CR2 RELOAD LL_I2C_HandleTransfer • CR2 NBYTES LL_I2C_HandleTransfer • CR2 AUTOEND LL_I2C_HandleTransfer • CR2 HEAD10R LL_I2C_HandleTransfer

LL_I2C_GetTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetTransferDirection (I2C_TypeDef * I2Cx)**
Function description	Indicate the value of transfer direction (slave mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_DIRECTION_WRITE – LL_I2C_DIRECTION_READ
Notes	• RESET: Write transfer, Slave enters in receiver mode. SET: Read transfer, Slave enters in transmitter mode.
Reference Manual to LL API cross reference:	• ISR DIR LL_I2C_GetTransferDirection

LL_I2C_GetAddressMatchCode

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetAddressMatchCode (I2C_TypeDef * I2Cx)**
Function description	Return the slave matched address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x3F
Reference Manual to LL API cross reference:	• ISR ADDCODE LL_I2C_GetAddressMatchCode

LL_I2C_EnableSMBusPECCompare

Function name	**__STATIC_INLINE void LL_I2C_EnableSMBusPECCompare (I2C_TypeDef * I2Cx)**
Function description	Enable internal comparison of the SMBus Packet Error byte (transmission or reception mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • This feature is cleared by hardware when the PEC byte is transferred, or when a STOP condition or an Address Matched is received. This bit has no effect when RELOAD bit is set. This bit has no effect in device mode when SBC bit is not set.
Reference Manual to LL API cross reference:	• CR2 PECBYTE LL_I2C_EnableSMBusPECCompare

LL_I2C_IsEnabledSMBusPECCompare

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCompare (I2C_TypeDef * I2Cx)**
Function description	Check if the SMBus Packet Error byte internal comparison is requested or not.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR2 PECBYTE LL_I2C_IsEnabledSMBusPECCompare

LL_I2C_GetSMBusPEC

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetSMBusPEC (I2C_TypeDef * I2Cx)**
Function description	Get the SMBus Packet Error byte calculated.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• PECR PEC LL_I2C_GetSMBusPEC

LL_I2C_ReceiveData8

Function name	**__STATIC_INLINE uint8_t LL_I2C_ReceiveData8 (I2C_TypeDef * I2Cx)**
Function description	Read Receive Data register.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• RXDR RXDATA LL_I2C_ReceiveData8

LL_I2C_TransmitData8

Function name	**__STATIC_INLINE void LL_I2C_TransmitData8 (I2C_TypeDef * I2Cx, uint8_t Data)**
Function description	Write in Transmit Data Register .
Parameters	• **I2Cx:** I2C Instance.

	• **Data:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• TXDR TXDATA LL_I2C_TransmitData8

LL_I2C_Init

Function name	**uint32_t LL_I2C_Init (I2C_TypeDef * I2Cx, LL_I2C_InitTypeDef * I2C_InitStruct)**
Function description	Initialize the I2C registers according to the specified parameters in I2C_InitStruct.
Parameters	• **I2Cx:** I2C Instance. • **I2C_InitStruct:** pointer to a LL_I2C_InitTypeDef structure.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: I2C registers are initialized – ERROR: Not applicable

LL_I2C_DeInit

Function name	**uint32_t LL_I2C_DeInit (I2C_TypeDef * I2Cx)**
Function description	De-initialize the I2C registers to their default reset values.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: I2C registers are de-initialized – ERROR: I2C registers are not de-initialized

LL_I2C_StructInit

Function name	**void LL_I2C_StructInit (LL_I2C_InitTypeDef * I2C_InitStruct)**
Function description	Set each LL_I2C_InitTypeDef field to default value.
Parameters	• **I2C_InitStruct:** Pointer to a LL_I2C_InitTypeDef structure.
Return values	• **None:**

85.3 I2C Firmware driver defines

85.3.1 I2C

Master Addressing Mode

LL_I2C_ADDRESSING_MODE_7BIT	Master operates in 7-bit addressing mode.
LL_I2C_ADDRESSING_MODE_10BIT	Master operates in 10-bit addressing mode.

Slave Address Length

LL_I2C_ADDRSLAVE_7BIT	Slave Address in 7-bit.
LL_I2C_ADDRSLAVE_10BIT	Slave Address in 10-bit.

Analog Filter Selection

LL_I2C_ANALOGFILTER_ENABLE	Analog filter is enabled.
LL_I2C_ANALOGFILTER_DISABLE	Analog filter is disabled.

Clear Flags Defines

LL_I2C_ICR_ADDRCF	Address Matched flag
LL_I2C_ICR_NACKCF	Not Acknowledge flag
LL_I2C_ICR_STOPCF	Stop detection flag
LL_I2C_ICR_BERRCF	Bus error flag
LL_I2C_ICR_ARLOCF	Arbitration Lost flag
LL_I2C_ICR_OVRCF	Overrun/Underrun flag
LL_I2C_ICR_PECCF	PEC error flag
LL_I2C_ICR_TIMOUTCF	Timeout detection flag
LL_I2C_ICR_ALERTCF	Alert flag

Read Write Direction

LL_I2C_DIRECTION_WRITE	Write transfer request by master, slave enters receiver mode.
LL_I2C_DIRECTION_READ	Read transfer request by master, slave enters transmitter mode.

DMA Register Data

LL_I2C_DMA_REG_DATA_TRANSMIT	Get address of data register used for transmission
LL_I2C_DMA_REG_DATA_RECEIVE	Get address of data register used for reception

Start And Stop Generation

LL_I2C_GENERATE_NOSTARTSTOP	Don't Generate Stop and Start condition.
LL_I2C_GENERATE_STOP	Generate Stop condition (Size should be set to 0).
LL_I2C_GENERATE_START_READ	Generate Start for read request.
LL_I2C_GENERATE_START_WRITE	Generate Start for write request.
LL_I2C_GENERATE_RESTART_7BIT_READ	Generate Restart for read request, slave 7Bit address.
LL_I2C_GENERATE_RESTART_7BIT_WRITE	Generate Restart for write request, slave 7Bit address.
LL_I2C_GENERATE_RESTART_10BIT_READ	Generate Restart for read request, slave 10Bit address.
LL_I2C_GENERATE_RESTART_10BIT_WRITE	Generate Restart for write request, slave 10Bit address.

Get Flags Defines

LL_I2C_ISR_TXE	Transmit data register empty
LL_I2C_ISR_TXIS	Transmit interrupt status

LL_I2C_ISR_RXNE	Receive data register not empty
LL_I2C_ISR_ADDR	Address matched (slave mode)
LL_I2C_ISR_NACKF	Not Acknowledge received flag
LL_I2C_ISR_STOPF	Stop detection flag
LL_I2C_ISR_TC	Transfer Complete (master mode)
LL_I2C_ISR_TCR	Transfer Complete Reload
LL_I2C_ISR_BERR	Bus error
LL_I2C_ISR_ARLO	Arbitration lost
LL_I2C_ISR_OVR	Overrun/Underrun (slave mode)
LL_I2C_ISR_PECERR	PEC Error in reception (SMBus mode)
LL_I2C_ISR_TIMEOUT	Timeout detection flag (SMBus mode)
LL_I2C_ISR_ALERT	SMBus alert (SMBus mode)
LL_I2C_ISR_BUSY	Bus busy

Acknowledge Generation

LL_I2C_ACK	ACK is sent after current received byte.
LL_I2C_NACK	NACK is sent after current received byte.

IT Defines

LL_I2C_CR1_TXIE	TX Interrupt enable
LL_I2C_CR1_RXIE	RX Interrupt enable
LL_I2C_CR1_ADDRIE	Address match Interrupt enable (slave only)
LL_I2C_CR1_NACKIE	Not acknowledge received Interrupt enable
LL_I2C_CR1_STOPIE	STOP detection Interrupt enable
LL_I2C_CR1_TCIE	Transfer Complete interrupt enable
LL_I2C_CR1_ERRIE	Error interrupts enable

Transfer End Mode

LL_I2C_MODE_RELOAD	Enable I2C Reload mode.
LL_I2C_MODE_AUTOEND	Enable I2C Automatic end mode with no HW PEC comparison.
LL_I2C_MODE_SOFTEND	Enable I2C Software end mode with no HW PEC comparison.
LL_I2C_MODE_SMBUS_RELOAD	Enable SMBUS Automatic end mode with HW PEC comparison.
LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC	Enable SMBUS Automatic end mode with HW PEC comparison.
LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC	Enable SMBUS Software end mode with HW PEC comparison.
LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC	Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC	Enable SMBUS Software end mode with HW PEC comparison.

Own Address 1 Length

LL_I2C_OWNADDRESS1_7BIT	Own address 1 is a 7-bit address.
LL_I2C_OWNADDRESS1_10BIT	Own address 1 is a 10-bit address.

Own Address 2 Masks

LL_I2C_OWNADDRESS2_NOMASK	Own Address2 No mask.
LL_I2C_OWNADDRESS2_MASK01	Only Address2 bits[7:2] are compared.
LL_I2C_OWNADDRESS2_MASK02	Only Address2 bits[7:3] are compared.
LL_I2C_OWNADDRESS2_MASK03	Only Address2 bits[7:4] are compared.
LL_I2C_OWNADDRESS2_MASK04	Only Address2 bits[7:5] are compared.
LL_I2C_OWNADDRESS2_MASK05	Only Address2 bits[7:6] are compared.
LL_I2C_OWNADDRESS2_MASK06	Only Address2 bits[7] are compared.
LL_I2C_OWNADDRESS2_MASK07	No comparison is done. All Address2 are acknowledged.

Peripheral Mode

LL_I2C_MODE_I2C	I2C Master or Slave mode
LL_I2C_MODE_SMBUS_HOST	SMBus Host address acknowledge
LL_I2C_MODE_SMBUS_DEVICE	SMBus Device default mode (Default address not acknowledge)
LL_I2C_MODE_SMBUS_DEVICE_ARP	SMBus Device Default address acknowledge

Transfer Request Direction

LL_I2C_REQUEST_WRITE	Master request a write transfer.
LL_I2C_REQUEST_READ	Master request a read transfer.

SMBus TimeoutA Mode SCL SDA Timeout

LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW	TimeoutA is used to detect SCL low level timeout.
LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH	TimeoutA is used to detect both SCL and SDA high level timeout.

SMBus Timeout Selection

LL_I2C_SMBUS_TIMEOUTA	TimeoutA enable bit
LL_I2C_SMBUS_TIMEOUTB	TimeoutB (extended clock) enable bit
LL_I2C_SMBUS_ALL_TIMEOUT	TimeoutA and TimeoutB (extended clock) enable bits

Convert SDA SCL timings

__LL_I2C_CONVERT_TIMINGS

Description:

- Configure the SDA setup, hold time and the SCL high, low period.

Parameters:

- __PRESCALER__: This parameter must be a value between Min_Data=0 and Max_Data=0xF.
- __DATA_SETUP_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tscldel = (SCLDEL+1)xtpresc)
- __DATA_HOLD_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tsdadel = SDADELxtpresc)
- __CLOCK_HIGH_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tsclh = (SCLH+1)xtpresc)
- __CLOCK_LOW_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tscll = (SCLL+1)xtpresc)

Return value:

- Value: between Min_Data=0 and Max_Data=0xFFFFFFFF

Common Write and read registers Macros

LL_I2C_WriteReg

Description:

- Write a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2C_ReadReg

Description:

- Read a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be read

Return value:

- Register: value

86 LL IWDG Generic Driver

86.1 IWDG Firmware driver API description

86.1.1 Detailed description of functions

LL_IWDG_Enable

Function name	**__STATIC_INLINE void LL_IWDG_Enable (IWDG_TypeDef * IWDGx)**
Function description	Start the Independent Watchdog.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None:**
Notes	• Except if the hardware watchdog option is selected
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_Enable

LL_IWDG_ReloadCounter

Function name	**__STATIC_INLINE void LL_IWDG_ReloadCounter (IWDG_TypeDef * IWDGx)**
Function description	Reloads IWDG counter with value defined in the reload register.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_ReloadCounter

LL_IWDG_EnableWriteAccess

Function name	**__STATIC_INLINE void LL_IWDG_EnableWriteAccess (IWDG_TypeDef * IWDGx)**
Function description	Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_EnableWriteAccess

LL_IWDG_DisableWriteAccess

Function name	**__STATIC_INLINE void LL_IWDG_DisableWriteAccess**

	(IWDG_TypeDef * IWDGx)
Function description	Disable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_DisableWriteAccess

LL_IWDG_SetPrescaler

Function name	**__STATIC_INLINE void LL_IWDG_SetPrescaler (IWDG_TypeDef * IWDGx, uint32_t Prescaler)**
Function description	Select the prescaler of the IWDG.
Parameters	• **IWDGx:** IWDG Instance • **Prescaler:** This parameter can be one of the following values: – LL_IWDG_PRESCALER_4 – LL_IWDG_PRESCALER_8 – LL_IWDG_PRESCALER_16 – LL_IWDG_PRESCALER_32 – LL_IWDG_PRESCALER_64 – LL_IWDG_PRESCALER_128 – LL_IWDG_PRESCALER_256
Return values	• **None:**
Reference Manual to LL API cross reference:	• PR PR LL_IWDG_SetPrescaler

LL_IWDG_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_IWDG_GetPrescaler (IWDG_TypeDef * IWDGx)**
Function description	Get the selected prescaler of the IWDG.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **Returned:** value can be one of the following values: – LL_IWDG_PRESCALER_4 – LL_IWDG_PRESCALER_8 – LL_IWDG_PRESCALER_16 – LL_IWDG_PRESCALER_32 – LL_IWDG_PRESCALER_64 – LL_IWDG_PRESCALER_128 – LL_IWDG_PRESCALER_256
Reference Manual to LL API cross reference:	• PR PR LL_IWDG_GetPrescaler

LL_IWDG_SetReloadCounter

Function name	**__STATIC_INLINE void LL_IWDG_SetReloadCounter (IWDG_TypeDef * IWDGx, uint32_t Counter)**
Function description	Specify the IWDG down-counter reload value.
Parameters	• **IWDGx:** IWDG Instance • **Counter:** Value between Min_Data=0 and Max_Data=0x0FFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• RLR RL LL_IWDG_SetReloadCounter

LL_IWDG_GetReloadCounter

Function name	**__STATIC_INLINE uint32_t LL_IWDG_GetReloadCounter (IWDG_TypeDef * IWDGx)**
Function description	Get the specified IWDG down-counter reload value.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **Value:** between Min_Data=0 and Max_Data=0x0FFF
Reference Manual to LL API cross reference:	• RLR RL LL_IWDG_GetReloadCounter

LL_IWDG_SetWindow

Function name	**__STATIC_INLINE void LL_IWDG_SetWindow (IWDG_TypeDef * IWDGx, uint32_t Window)**
Function description	Specify high limit of the window value to be compared to the down-counter.
Parameters	• **IWDGx:** IWDG Instance • **Window:** Value between Min_Data=0 and Max_Data=0x0FFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• WINR WIN LL_IWDG_SetWindow

LL_IWDG_GetWindow

Function name	**__STATIC_INLINE uint32_t LL_IWDG_GetWindow (IWDG_TypeDef * IWDGx)**
Function description	Get the high limit of the window value specified.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **Value:** between Min_Data=0 and Max_Data=0x0FFF
Reference Manual to	• WINR WIN LL_IWDG_GetWindow

LL API cross reference:

LL_IWDG_IsActiveFlag_PVU

Function name	**__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_PVU (IWDG_TypeDef * IWDGx)**
Function description	Check if flag Prescaler Value Update is set or not.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR PVU LL_IWDG_IsActiveFlag_PVU

LL_IWDG_IsActiveFlag_RVU

Function name	**__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_RVU (IWDG_TypeDef * IWDGx)**
Function description	Check if flag Reload Value Update is set or not.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR RVU LL_IWDG_IsActiveFlag_RVU

LL_IWDG_IsActiveFlag_WVU

Function name	**__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_WVU (IWDG_TypeDef * IWDGx)**
Function description	Check if flag Window Value Update is set or not.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR WVU LL_IWDG_IsActiveFlag_WVU

LL_IWDG_IsReady

Function name	**__STATIC_INLINE uint32_t LL_IWDG_IsReady (IWDG_TypeDef * IWDGx)**
Function description	Check if all flags Prescaler, Reload & Window Value Update are reset or not.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **State:** of bits (1 or 0).
Reference Manual to LL API cross	• SR PVU LL_IWDG_IsReady • SR WVU LL_IWDG_IsReady

reference:
- SR RVU LL_IWDG_IsReady

86.2 IWDG Firmware driver defines

86.2.1 IWDG

Get Flags Defines

LL_IWDG_SR_PVU	Watchdog prescaler value update
LL_IWDG_SR_RVU	Watchdog counter reload value update
LL_IWDG_SR_WVU	Watchdog counter window value update

Prescaler Divider

LL_IWDG_PRESCALER_4	Divider by 4
LL_IWDG_PRESCALER_8	Divider by 8
LL_IWDG_PRESCALER_16	Divider by 16
LL_IWDG_PRESCALER_32	Divider by 32
LL_IWDG_PRESCALER_64	Divider by 64
LL_IWDG_PRESCALER_128	Divider by 128
LL_IWDG_PRESCALER_256	Divider by 256

Common Write and read registers Macros

LL_IWDG_WriteReg

Description:

- Write a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_IWDG_ReadReg

Description:

- Read a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

87 LL LPTIM Generic Driver

87.1 LPTIM Firmware driver registers structures

87.1.1 LL_LPTIM_InitTypeDef

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t Prescaler***
- ***uint32_t Waveform***
- ***uint32_t Polarity***

Field Documentation

- ***uint32_t LL_LPTIM_InitTypeDef::ClockSource***
  Specifies the source of the clock used by the LPTIM instance. This parameter can be a value of ***LPTIM_LL_EC_CLK_SOURCE***.This feature can be modified afterwards using unitary function **LL_LPTIM_SetClockSource()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Prescaler***
  Specifies the prescaler division ratio. This parameter can be a value of ***LPTIM_LL_EC_PRESCALER***.This feature can be modified afterwards using using unitary function **LL_LPTIM_SetPrescaler()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Waveform***
  Specifies the waveform shape. This parameter can be a value of ***LPTIM_LL_EC_OUTPUT_WAVEFORM***.This feature can be modified afterwards using unitary function **LL_LPTIM_ConfigOutput()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Polarity***
  Specifies waveform polarity. This parameter can be a value of ***LPTIM_LL_EC_OUTPUT_POLARITY***.This feature can be modified afterwards using unitary function **LL_LPTIM_ConfigOutput()**.

87.2 LPTIM Firmware driver API description

87.2.1 Detailed description of functions

LL_LPTIM_Enable

Function name	**__STATIC_INLINE void LL_LPTIM_Enable (LPTIM_TypeDef * LPTIMx)**
Function description	Enable the LPTIM instance.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Notes	• After setting the ENABLE bit, a delay of two counter clock is needed before the LPTIM instance is actually enabled.
Reference Manual to LL API cross reference:	• CR ENABLE LL_LPTIM_Enable

LL_LPTIM_Disable

Function name	**__STATIC_INLINE void LL_LPTIM_Disable (LPTIM_TypeDef * LPTIMx)**
Function description	Disable the LPTIM instance.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR ENABLE LL_LPTIM_Disable

LL_LPTIM_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabled (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the LPTIM instance is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR ENABLE LL_LPTIM_IsEnabled

LL_LPTIM_StartCounter

Function name	**__STATIC_INLINE void LL_LPTIM_StartCounter (LPTIM_TypeDef * LPTIMx, uint32_t OperatingMode)**
Function description	Starts the LPTIM counter in the desired mode.
Parameters	• **LPTIMx:** Low-Power Timer instance • **OperatingMode:** This parameter can be one of the following values: – LL_LPTIM_OPERATING_MODE_CONTINUOUS – LL_LPTIM_OPERATING_MODE_ONESHOT
Return values	• **None:**
Notes	• LPTIM instance must be enabled before starting the counter. • It is possible to change on the fly from One Shot mode to Continuous mode.
Reference Manual to LL API cross reference:	• CR CNTSTRT LL_LPTIM_StartCounter • CR SNGSTRT LL_LPTIM_StartCounter

LL_LPTIM_SetUpdateMode

Function name	**__STATIC_INLINE void LL_LPTIM_SetUpdateMode (LPTIM_TypeDef * LPTIMx, uint32_t UpdateMode)**
Function description	Set the LPTIM registers update mode (enable/disable register preload)

Parameters	• **LPTIMx:** Low-Power Timer instance • **UpdateMode:** This parameter can be one of the following values: – LL_LPTIM_UPDATE_MODE_IMMEDIATE – LL_LPTIM_UPDATE_MODE_ENDOFPERIOD
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR PRELOAD LL_LPTIM_SetUpdateMode

LL_LPTIM_GetUpdateMode

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetUpdateMode (LPTIM_TypeDef * LPTIMx)**
Function description	Get the LPTIM registers update mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_UPDATE_MODE_IMMEDIATE – LL_LPTIM_UPDATE_MODE_ENDOFPERIOD
Reference Manual to LL API cross reference:	• CFGR PRELOAD LL_LPTIM_GetUpdateMode

LL_LPTIM_SetAutoReload

Function name	**__STATIC_INLINE void LL_LPTIM_SetAutoReload (LPTIM_TypeDef * LPTIMx, uint32_t AutoReload)**
Function description	Set the auto reload value.
Parameters	• **LPTIMx:** Low-Power Timer instance • **AutoReload:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Return values	• **None:**
Notes	• The LPTIMx_ARR register content must only be modified when the LPTIM is enabled • After a write to the LPTIMx_ARR register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the ARROK flag be set, will lead to unpredictable results. • autoreload value be strictly greater than the compare value.
Reference Manual to LL API cross reference:	• ARR ARR LL_LPTIM_SetAutoReload

LL_LPTIM_GetAutoReload

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetAutoReload (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual auto reload value.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **AutoReload:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• ARR ARR LL_LPTIM_GetAutoReload

LL_LPTIM_SetCompare

Function name	**__STATIC_INLINE void LL_LPTIM_SetCompare (LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)**
Function description	Set the compare value.
Parameters	• **LPTIMx:** Low-Power Timer instance • **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Return values	• **None:**
Notes	• After a write to the LPTIMx_CMP register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMPOK flag be set, will lead to unpredictable results.
Reference Manual to LL API cross reference:	• CMP CMP LL_LPTIM_SetCompare

LL_LPTIM_GetCompare

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetCompare (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual compare value.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• CMP CMP LL_LPTIM_GetCompare

LL_LPTIM_GetCounter

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetCounter (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual counter value.

Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Counter:** value
Notes	• When the LPTIM instance is running with an asynchronous clock, reading the LPTIMx_CNT register may return unreliable values. So in this case it is necessary to perform two consecutive read accesses and verify that the two returned values are identical.
Reference Manual to LL API cross reference:	• CNT CNT LL_LPTIM_GetCounter

LL_LPTIM_SetCounterMode

Function name	**__STATIC_INLINE void LL_LPTIM_SetCounterMode (LPTIM_TypeDef * LPTIMx, uint32_t CounterMode)**
Function description	Set the counter mode (selection of the LPTIM counter clock source).
Parameters	• **LPTIMx:** Low-Power Timer instance • **CounterMode:** This parameter can be one of the following values: – LL_LPTIM_COUNTER_MODE_INTERNAL – LL_LPTIM_COUNTER_MODE_EXTERNAL
Return values	• **None:**
Notes	• The counter mode can be set only when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR COUNTMODE LL_LPTIM_SetCounterMode

LL_LPTIM_GetCounterMode

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetCounterMode (LPTIM_TypeDef * LPTIMx)**
Function description	Get the counter mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_COUNTER_MODE_INTERNAL – LL_LPTIM_COUNTER_MODE_EXTERNAL
Reference Manual to LL API cross reference:	• CFGR COUNTMODE LL_LPTIM_GetCounterMode

LL_LPTIM_ConfigOutput

Function name	**__STATIC_INLINE void LL_LPTIM_ConfigOutput (LPTIM_TypeDef * LPTIMx, uint32_t Waveform, uint32_t Polarity)**

Function description	Configure the LPTIM instance output (LPTIMx_OUT)
Parameters	• **LPTIMx:** Low-Power Timer instance • **Waveform:** This parameter can be one of the following values: – LL_LPTIM_OUTPUT_WAVEFORM_PWM – LL_LPTIM_OUTPUT_WAVEFORM_SETONCE • **Polarity:** This parameter can be one of the following values: – LL_LPTIM_OUTPUT_POLARITY_REGULAR – LL_LPTIM_OUTPUT_POLARITY_INVERSE
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled. • Regarding the LPTIM output polarity the change takes effect immediately, so the output default value will change immediately after the polarity is re-configured, even before the timer is enabled.
Reference Manual to LL API cross reference:	• CFGR WAVE LL_LPTIM_ConfigOutput • CFGR WAVPOL LL_LPTIM_ConfigOutput

LL_LPTIM_SetWaveform

Function name	**__STATIC_INLINE void LL_LPTIM_SetWaveform (LPTIM_TypeDef * LPTIMx, uint32_t Waveform)**
Function description	Set waveform shape.
Parameters	• **LPTIMx:** Low-Power Timer instance • **Waveform:** This parameter can be one of the following values: – LL_LPTIM_OUTPUT_WAVEFORM_PWM – LL_LPTIM_OUTPUT_WAVEFORM_SETONCE
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR WAVE LL_LPTIM_SetWaveform

LL_LPTIM_GetWaveform

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetWaveform (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual waveform shape.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_OUTPUT_WAVEFORM_PWM – LL_LPTIM_OUTPUT_WAVEFORM_SETONCE
Reference Manual to LL API cross reference:	• CFGR WAVE LL_LPTIM_GetWaveform

LL_LPTIM_SetPolarity

Function name	**__STATIC_INLINE void LL_LPTIM_SetPolarity (LPTIM_TypeDef * LPTIMx, uint32_t Polarity)**
Function description	Set output polarity.
Parameters	• **LPTIMx:** Low-Power Timer instance • **Polarity:** This parameter can be one of the following values: – LL_LPTIM_OUTPUT_POLARITY_REGULAR – LL_LPTIM_OUTPUT_POLARITY_INVERSE
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR WAVPOL LL_LPTIM_SetPolarity

LL_LPTIM_GetPolarity

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetPolarity (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual output polarity.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_OUTPUT_POLARITY_REGULAR – LL_LPTIM_OUTPUT_POLARITY_INVERSE
Reference Manual to LL API cross reference:	• CFGR WAVPOL LL_LPTIM_GetPolarity

LL_LPTIM_SetPrescaler

Function name	**__STATIC_INLINE void LL_LPTIM_SetPrescaler (LPTIM_TypeDef * LPTIMx, uint32_t Prescaler)**
Function description	Set actual prescaler division ratio.
Parameters	• **LPTIMx:** Low-Power Timer instance • **Prescaler:** This parameter can be one of the following values: – LL_LPTIM_PRESCALER_DIV1 – LL_LPTIM_PRESCALER_DIV2 – LL_LPTIM_PRESCALER_DIV4 – LL_LPTIM_PRESCALER_DIV8 – LL_LPTIM_PRESCALER_DIV16 – LL_LPTIM_PRESCALER_DIV32 – LL_LPTIM_PRESCALER_DIV64 – LL_LPTIM_PRESCALER_DIV128
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled. • When the LPTIM is configured to be clocked by an internal clock source and the LPTIM counter is configured to be

	updated by active edges detected on the LPTIM external Input1, the internal clock provided to the LPTIM must be not be prescaled.
Reference Manual to LL API cross reference:	• CFGR PRESC LL_LPTIM_SetPrescaler

LL_LPTIM_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetPrescaler (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual prescaler division ratio.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_PRESCALER_DIV1 – LL_LPTIM_PRESCALER_DIV2 – LL_LPTIM_PRESCALER_DIV4 – LL_LPTIM_PRESCALER_DIV8 – LL_LPTIM_PRESCALER_DIV16 – LL_LPTIM_PRESCALER_DIV32 – LL_LPTIM_PRESCALER_DIV64 – LL_LPTIM_PRESCALER_DIV128
Reference Manual to LL API cross reference:	• CFGR PRESC LL_LPTIM_GetPrescaler

LL_LPTIM_SetInput1Src

Function name	**__STATIC_INLINE void LL_LPTIM_SetInput1Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)**
Function description	Set LPTIM input 1 source (default GPIO).
Parameters	• **LPTIMx:** Low-Power Timer instance • **Src:** This parameter can be one of the following values: – LL_LPTIM_INPUT1_SRC_GPIO – LL_LPTIM_INPUT1_SRC_COMP1 – LL_LPTIM_INPUT1_SRC_COMP2 – LL_LPTIM_INPUT1_SRC_COMP1_COMP2
Return values	• **None:**
Reference Manual to LL API cross reference:	• OR OR_0 LL_LPTIM_SetInput1Src • OR OR_1 LL_LPTIM_SetInput1Src

LL_LPTIM_SetInput2Src

Function name	**__STATIC_INLINE void LL_LPTIM_SetInput2Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)**
Function description	Set LPTIM input 2 source (default GPIO).
Parameters	• **LPTIMx:** Low-Power Timer instance

	• **Src:** This parameter can be one of the following values: – LL_LPTIM_INPUT2_SRC_GPIO – LL_LPTIM_INPUT2_SRC_COMP2
Return values	• **None:**
Reference Manual to LL API cross reference:	• OR OR_0 LL_LPTIM_SetInput2Src

LL_LPTIM_EnableTimeout

Function name	**__STATIC_INLINE void LL_LPTIM_EnableTimeout (LPTIM_TypeDef * LPTIMx)**
Function description	Enable the timeout function.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled. • The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. • The timeout value corresponds to the compare value; if no trigger occurs within the expected time frame, the MCU is waked-up by the compare match event.
Reference Manual to LL API cross reference:	• CFGR TIMOUT LL_LPTIM_EnableTimeout

LL_LPTIM_DisableTimeout

Function name	**__STATIC_INLINE void LL_LPTIM_DisableTimeout (LPTIM_TypeDef * LPTIMx)**
Function description	Disable the timeout function.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled. • A trigger event arriving when the timer is already started will be ignored.
Reference Manual to LL API cross reference:	• CFGR TIMOUT LL_LPTIM_DisableTimeout

LL_LPTIM_IsEnabledTimeout

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledTimeout (LPTIM_TypeDef * LPTIMx)**
Function description	Indicate whether the timeout function is enabled.

Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CFGR TIMOUT LL_LPTIM_IsEnabledTimeout

LL_LPTIM_TrigSw

Function name	**__STATIC_INLINE void LL_LPTIM_TrigSw (LPTIM_TypeDef * LPTIMx)**
Function description	Start the LPTIM counter.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR TRIGEN LL_LPTIM_TrigSw

LL_LPTIM_ConfigTrigger

Function name	**__STATIC_INLINE void LL_LPTIM_ConfigTrigger (LPTIM_TypeDef * LPTIMx, uint32_t Source, uint32_t Filter, uint32_t Polarity)**
Function description	Configure the external trigger used as a trigger event for the LPTIM.
Parameters	• **LPTIMx:** Low-Power Timer instance • **Source:** This parameter can be one of the following values: – LL_LPTIM_TRIG_SOURCE_GPIO – LL_LPTIM_TRIG_SOURCE_RTCALARMA – LL_LPTIM_TRIG_SOURCE_RTCALARMB – LL_LPTIM_TRIG_SOURCE_RTCTAMP1 – LL_LPTIM_TRIG_SOURCE_RTCTAMP2 – LL_LPTIM_TRIG_SOURCE_RTCTAMP3 – LL_LPTIM_TRIG_SOURCE_COMP1 – LL_LPTIM_TRIG_SOURCE_COMP2 • **Filter:** This parameter can be one of the following values: – LL_LPTIM_TRIG_FILTER_NONE – LL_LPTIM_TRIG_FILTER_2 – LL_LPTIM_TRIG_FILTER_4 – LL_LPTIM_TRIG_FILTER_8 • **Polarity:** This parameter can be one of the following values: – LL_LPTIM_TRIG_POLARITY_RISING – LL_LPTIM_TRIG_POLARITY_FALLING – LL_LPTIM_TRIG_POLARITY_RISING_FALLING
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled.

- An internal clock source must be present when a digital filter is required for the trigger.

Reference Manual to LL API cross reference:
- CFGR TRIGSEL LL_LPTIM_ConfigTrigger
- CFGR TRGFLT LL_LPTIM_ConfigTrigger
- CFGR TRIGEN LL_LPTIM_ConfigTrigger

LL_LPTIM_GetTriggerSource

Function name: **__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerSource (LPTIM_TypeDef * LPTIMx)**

Function description: Get actual external trigger source.

Parameters:
- **LPTIMx:** Low-Power Timer instance

Return values:
- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP3
  - LL_LPTIM_TRIG_SOURCE_COMP1
  - LL_LPTIM_TRIG_SOURCE_COMP2

Reference Manual to LL API cross reference:
- CFGR TRIGSEL LL_LPTIM_GetTriggerSource

LL_LPTIM_GetTriggerFilter

Function name: **__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerFilter (LPTIM_TypeDef * LPTIMx)**

Function description: Get actual external trigger filter.

Parameters:
- **LPTIMx:** Low-Power Timer instance

Return values:
- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8

Reference Manual to LL API cross reference:
- CFGR TRGFLT LL_LPTIM_GetTriggerFilter

LL_LPTIM_GetTriggerPolarity

Function name: **__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerPolarity (LPTIM_TypeDef * LPTIMx)**

Function description: Get actual external trigger polarity.

Parameters:
- **LPTIMx:** Low-Power Timer instance

Return values:
- **Returned:** value can be one of the following values:

- LL_LPTIM_TRIG_POLARITY_RISING
- LL_LPTIM_TRIG_POLARITY_FALLING
- LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:	• CFGR TRIGEN LL_LPTIM_GetTriggerPolarity

LL_LPTIM_SetClockSource

Function name	**__STATIC_INLINE void LL_LPTIM_SetClockSource (LPTIM_TypeDef * LPTIMx, uint32_t ClockSource)**
Function description	Set the source of the clock used by the LPTIM instance.
Parameters	• **LPTIMx:** Low-Power Timer instance • **ClockSource:** This parameter can be one of the following values: – LL_LPTIM_CLK_SOURCE_INTERNAL – LL_LPTIM_CLK_SOURCE_EXTERNAL
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR CKSEL LL_LPTIM_SetClockSource

LL_LPTIM_GetClockSource

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetClockSource (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual LPTIM instance clock source.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_CLK_SOURCE_INTERNAL – LL_LPTIM_CLK_SOURCE_EXTERNAL
Reference Manual to LL API cross reference:	• CFGR CKSEL LL_LPTIM_GetClockSource

LL_LPTIM_ConfigClock

Function name	**__STATIC_INLINE void LL_LPTIM_ConfigClock (LPTIM_TypeDef * LPTIMx, uint32_t ClockFilter, uint32_t ClockPolarity)**
Function description	Configure the active edge or edges used by the counter when the LPTIM is clocked by an external clock source.
Parameters	• **LPTIMx:** Low-Power Timer instance • **ClockFilter:** This parameter can be one of the following values: – LL_LPTIM_CLK_FILTER_NONE

	– LL_LPTIM_CLK_FILTER_2 – LL_LPTIM_CLK_FILTER_4 – LL_LPTIM_CLK_FILTER_8 • **ClockPolarity:** This parameter can be one of the following values: – LL_LPTIM_CLK_POLARITY_RISING – LL_LPTIM_CLK_POLARITY_FALLING – LL_LPTIM_CLK_POLARITY_RISING_FALLING
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled. • When both external clock signal edges are considered active ones, the LPTIM must also be clocked by an internal clock source with a frequency equal to at least four times the external clock frequency. • An internal clock source must be present when a digital filter is required for external clock.
Reference Manual to LL API cross reference:	• CFGR CKFLT LL_LPTIM_ConfigClock • CFGR CKPOL LL_LPTIM_ConfigClock

LL_LPTIM_GetClockPolarity

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetClockPolarity (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual clock polarity.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_CLK_POLARITY_RISING – LL_LPTIM_CLK_POLARITY_FALLING – LL_LPTIM_CLK_POLARITY_RISING_FALLING
Reference Manual to LL API cross reference:	• CFGR CKPOL LL_LPTIM_GetClockPolarity

LL_LPTIM_GetClockFilter

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetClockFilter (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual clock digital filter.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_CLK_FILTER_NONE – LL_LPTIM_CLK_FILTER_2 – LL_LPTIM_CLK_FILTER_4 – LL_LPTIM_CLK_FILTER_8
Reference Manual to LL API cross	• CFGR CKFLT LL_LPTIM_GetClockFilter

reference:

LL_LPTIM_SetEncoderMode

Function name	**__STATIC_INLINE void LL_LPTIM_SetEncoderMode (LPTIM_TypeDef * LPTIMx, uint32_t EncoderMode)**
Function description	Configure the encoder mode.
Parameters	• **LPTIMx:** Low-Power Timer instance • **EncoderMode:** This parameter can be one of the following values: – LL_LPTIM_ENCODER_MODE_RISING – LL_LPTIM_ENCODER_MODE_FALLING – LL_LPTIM_ENCODER_MODE_RISING_FALLING
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR CKPOL LL_LPTIM_SetEncoderMode

LL_LPTIM_GetEncoderMode

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_GetEncoderMode (LPTIM_TypeDef * LPTIMx)**
Function description	Get actual encoder mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_LPTIM_ENCODER_MODE_RISING – LL_LPTIM_ENCODER_MODE_FALLING – LL_LPTIM_ENCODER_MODE_RISING_FALLING
Reference Manual to LL API cross reference:	• CFGR CKPOL LL_LPTIM_GetEncoderMode

LL_LPTIM_EnableEncoderMode

Function name	**__STATIC_INLINE void LL_LPTIM_EnableEncoderMode (LPTIM_TypeDef * LPTIMx)**
Function description	Enable the encoder mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled. • In this mode the LPTIM instance must be clocked by an internal clock source. Also, the prescaler division ratio must be equal to 1. • LPTIM instance must be configured in continuous mode prior

	enabling the encoder mode.
Reference Manual to LL API cross reference:	• CFGR ENC LL_LPTIM_EnableEncoderMode

LL_LPTIM_DisableEncoderMode

Function name	**__STATIC_INLINE void LL_LPTIM_DisableEncoderMode (LPTIM_TypeDef * LPTIMx)**
Function description	Disable the encoder mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Notes	• This function must be called when the LPTIM instance is disabled.
Reference Manual to LL API cross reference:	• CFGR ENC LL_LPTIM_DisableEncoderMode

LL_LPTIM_IsEnabledEncoderMode

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledEncoderMode (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the LPTIM operates in encoder mode.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CFGR ENC LL_LPTIM_IsEnabledEncoderMode

LL_LPTIM_ClearFLAG_CMPM

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFLAG_CMPM (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the compare match flag (CMPMCF)
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR CMPMCF LL_LPTIM_ClearFLAG_CMPM

LL_LPTIM_IsActiveFlag_CMPM

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMPM (LPTIM_TypeDef * LPTIMx)**
Function description	Inform application whether a compare match interrupt has occurred.

Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CMPM LL_LPTIM_IsActiveFlag_CMPM

LL_LPTIM_ClearFLAG_ARRM

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFLAG_ARRM (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the autoreload match flag (ARRMCF)
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR ARRMCF LL_LPTIM_ClearFLAG_ARRM

LL_LPTIM_IsActiveFlag_ARRM

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARRM (LPTIM_TypeDef * LPTIMx)**
Function description	Inform application whether a autoreload match interrupt has occured.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ARRM LL_LPTIM_IsActiveFlag_ARRM

LL_LPTIM_ClearFlag_EXTTRIG

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the external trigger valid edge flag(EXTTRIGCF).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR EXTTRIGCF LL_LPTIM_ClearFlag_EXTTRIG

LL_LPTIM_IsActiveFlag_EXTTRIG

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)**
Function description	Inform application whether a valid edge on the selected external trigger input has occurred.

Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR EXTTRIG LL_LPTIM_IsActiveFlag_EXTTRIG

LL_LPTIM_ClearFlag_CMPOK

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFlag_CMPOK (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the compare register update interrupt flag (CMPOKCF).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR CMPOKCF LL_LPTIM_ClearFlag_CMPOK

LL_LPTIM_IsActiveFlag_CMPOK

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMPOK (LPTIM_TypeDef * LPTIMx)**
Function description	Informs application whether the APB bus write operation to the LPTIMx_CMP register has been successfully completed; If so, a new one can be initiated.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CMPOK LL_LPTIM_IsActiveFlag_CMPOK

LL_LPTIM_ClearFlag_ARROK

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFlag_ARROK (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the autoreload register update interrupt flag (ARROKCF).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR ARROKCF LL_LPTIM_ClearFlag_ARROK

LL_LPTIM_IsActiveFlag_ARROK

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARROK (LPTIM_TypeDef * LPTIMx)**
Function description	Informs application whether the APB bus write operation to the

	LPTIMx_ARR register has been successfully completed; If so, a new one can be initiated.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ARROK LL_LPTIM_IsActiveFlag_ARROK

LL_LPTIM_ClearFlag_UP

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFlag_UP (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the counter direction change to up interrupt flag (UPCF).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR UPCF LL_LPTIM_ClearFlag_UP

LL_LPTIM_IsActiveFlag_UP

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UP (LPTIM_TypeDef * LPTIMx)**
Function description	Informs the application whether the counter direction has changed from down to up (when the LPTIM instance operates in encoder mode).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR UP LL_LPTIM_IsActiveFlag_UP

LL_LPTIM_ClearFlag_DOWN

Function name	**__STATIC_INLINE void LL_LPTIM_ClearFlag_DOWN (LPTIM_TypeDef * LPTIMx)**
Function description	Clear the counter direction change to down interrupt flag (DOWNCF).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR DOWNCF LL_LPTIM_ClearFlag_DOWN

LL_LPTIM_IsActiveFlag_DOWN

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DOWN (LPTIM_TypeDef * LPTIMx)**
Function description	Informs the application whether the counter direction has changed from up to down (when the LPTIM instance operates in encoder mode).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR DOWN LL_LPTIM_IsActiveFlag_DOWN

LL_LPTIM_EnableIT_CMPM

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_CMPM (LPTIM_TypeDef * LPTIMx)**
Function description	Enable compare match interrupt (CMPMIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER CMPMIE LL_LPTIM_EnableIT_CMPM

LL_LPTIM_DisableIT_CMPM

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_CMPM (LPTIM_TypeDef * LPTIMx)**
Function description	Disable compare match interrupt (CMPMIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER CMPMIE LL_LPTIM_DisableIT_CMPM

LL_LPTIM_IsEnabledIT_CMPM

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPM (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the compare match interrupt (CMPMIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER CMPMIE LL_LPTIM_IsEnabledIT_CMPM

LL_LPTIM_EnableIT_ARRM

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_ARRM (LPTIM_TypeDef * LPTIMx)**
Function description	Enable autoreload match interrupt (ARRMIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER ARRMIE LL_LPTIM_EnableIT_ARRM

LL_LPTIM_DisableIT_ARRM

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_ARRM (LPTIM_TypeDef * LPTIMx)**
Function description	Disable autoreload match interrupt (ARRMIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER ARRMIE LL_LPTIM_DisableIT_ARRM

LL_LPTIM_IsEnabledIT_ARRM

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARRM (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the autoreload match interrupt (ARRMIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER ARRMIE LL_LPTIM_IsEnabledIT_ARRM

LL_LPTIM_EnableIT_EXTTRIG

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)**
Function description	Enable external trigger valid edge interrupt (EXTTRIGIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER EXTTRIGIE LL_LPTIM_EnableIT_EXTTRIG

LL_LPTIM_DisableIT_EXTTRIG

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)**
Function description	Disable external trigger valid edge interrupt (EXTTRIGIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER EXTTRIGIE LL_LPTIM_DisableIT_EXTTRIG

LL_LPTIM_IsEnabledIT_EXTTRIG

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates external trigger valid edge interrupt (EXTTRIGIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER EXTTRIGIE LL_LPTIM_IsEnabledIT_EXTTRIG

LL_LPTIM_EnableIT_CMPOK

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_CMPOK (LPTIM_TypeDef * LPTIMx)**
Function description	Enable compare register write completed interrupt (CMPOKIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER CMPOKIE LL_LPTIM_EnableIT_CMPOK

LL_LPTIM_DisableIT_CMPOK

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_CMPOK (LPTIM_TypeDef * LPTIMx)**
Function description	Disable compare register write completed interrupt (CMPOKIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER CMPOKIE LL_LPTIM_DisableIT_CMPOK

LL_LPTIM_IsEnabledIT_CMPOK

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPOK (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the compare register write completed interrupt (CMPOKIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER CMPOKIE LL_LPTIM_IsEnabledIT_CMPOK

LL_LPTIM_EnableIT_ARROK

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_ARROK (LPTIM_TypeDef * LPTIMx)**
Function description	Enable autoreload register write completed interrupt (ARROKIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER ARROKIE LL_LPTIM_EnableIT_ARROK

LL_LPTIM_DisableIT_ARROK

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_ARROK (LPTIM_TypeDef * LPTIMx)**
Function description	Disable autoreload register write completed interrupt (ARROKIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER ARROKIE LL_LPTIM_DisableIT_ARROK

LL_LPTIM_IsEnabledIT_ARROK

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARROK (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the autoreload register write completed interrupt (ARROKIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER ARROKIE LL_LPTIM_IsEnabledIT_ARROK

LL_LPTIM_EnableIT_UP

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_UP (LPTIM_TypeDef * LPTIMx)**
Function description	Enable direction change to up interrupt (UPIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER UPIE LL_LPTIM_EnableIT_UP

LL_LPTIM_DisableIT_UP

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_UP (LPTIM_TypeDef * LPTIMx)**
Function description	Disable direction change to up interrupt (UPIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER UPIE LL_LPTIM_DisableIT_UP

LL_LPTIM_IsEnabledIT_UP

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UP (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the direction change to up interrupt (UPIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER UPIE LL_LPTIM_IsEnabledIT_UP

LL_LPTIM_EnableIT_DOWN

Function name	**__STATIC_INLINE void LL_LPTIM_EnableIT_DOWN (LPTIM_TypeDef * LPTIMx)**
Function description	Enable direction change to down interrupt (DOWNIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER DOWNIE LL_LPTIM_EnableIT_DOWN

LL_LPTIM_DisableIT_DOWN

Function name	**__STATIC_INLINE void LL_LPTIM_DisableIT_DOWN (LPTIM_TypeDef * LPTIMx)**
Function description	Disable direction change to down interrupt (DOWNIE).
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IER DOWNIE LL_LPTIM_DisableIT_DOWN

LL_LPTIM_IsEnabledIT_DOWN

Function name	**__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_DOWN (LPTIM_TypeDef * LPTIMx)**
Function description	Indicates whether the direction change to down interrupt (DOWNIE) is enabled.
Parameters	• **LPTIMx:** Low-Power Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• IER DOWNIE LL_LPTIM_IsEnabledIT_DOWN

LL_LPTIM_DeInit

Function name	**ErrorStatus LL_LPTIM_DeInit (LPTIM_TypeDef * LPTIMx)**
Function description	Set LPTIMx registers to their reset values.
Parameters	• **LPTIMx:** LP Timer instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: LPTIMx registers are de-initialized – ERROR: invalid LPTIMx instance

LL_LPTIM_StructInit

Function name	**void LL_LPTIM_StructInit (LL_LPTIM_InitTypeDef * LPTIM_InitStruct)**
Function description	Set each fields of the LPTIM_InitStruct structure to its default value.
Parameters	• **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure
Return values	• **None:**

LL_LPTIM_Init

Function name	**ErrorStatus LL_LPTIM_Init (LPTIM_TypeDef * LPTIMx, LL_LPTIM_InitTypeDef * LPTIM_InitStruct)**

Function description	Configure the LPTIMx peripheral according to the specified parameters.
Parameters	• **LPTIMx:** LP Timer Instance • **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: LPTIMx instance has been initialized – ERROR: LPTIMx instance hasn't been initialized
Notes	• LL_LPTIM_Init can only be called when the LPTIM instance is disabled. • LPTIMx can be disabled using unitary function LL_LPTIM_Disable().

87.3 LPTIM Firmware driver defines

87.3.1 LPTIM

Input1 Source

LL_LPTIM_INPUT1_SRC_GPIO	For LPTIM1 and LPTIM2
LL_LPTIM_INPUT1_SRC_COMP1	For LPTIM1 and LPTIM2
LL_LPTIM_INPUT1_SRC_COMP2	For LPTIM2
LL_LPTIM_INPUT1_SRC_COMP1_COMP2	For LPTIM2

Input2 Source

LL_LPTIM_INPUT2_SRC_GPIO	For LPTIM1
LL_LPTIM_INPUT2_SRC_COMP2	For LPTIM1

Clock Filter

LL_LPTIM_CLK_FILTER_NONE	Any external clock signal level change is considered as a valid transition
LL_LPTIM_CLK_FILTER_2	External clock signal level change must be stable for at least 2 clock periods before it is considered as valid transition
LL_LPTIM_CLK_FILTER_4	External clock signal level change must be stable for at least 4 clock periods before it is considered as valid transition
LL_LPTIM_CLK_FILTER_8	External clock signal level change must be stable for at least 8 clock periods before it is considered as valid transition

Clock Polarity

LL_LPTIM_CLK_POLARITY_RISING	The rising edge is the active edge used for counting
LL_LPTIM_CLK_POLARITY_FALLING	The falling edge is the active edge used for counting
LL_LPTIM_CLK_POLARITY_RISING_FALLING	Both edges are active edges

Clock Source

LL_LPTIM_CLK_SOURCE_INTERNAL	LPTIM is clocked by internal clock source (APB clock or any of the embedded oscillators)
LL_LPTIM_CLK_SOURCE_EXTERNAL	LPTIM is clocked by an external clock source through the LPTIM external Input1

Counter Mode

LL_LPTIM_COUNTER_MODE_INTERNAL	The counter is incremented following each internal clock pulse
LL_LPTIM_COUNTER_MODE_EXTERNAL	The counter is incremented following each valid clock pulse on the LPTIM external Input1

Encoder Mode

LL_LPTIM_ENCODER_MODE_RISING	The rising edge is the active edge used for counting
LL_LPTIM_ENCODER_MODE_FALLING	The falling edge is the active edge used for counting
LL_LPTIM_ENCODER_MODE_RISING_FALLING	Both edges are active edges

Get Flags Defines

LL_LPTIM_ISR_CMPM	Compare match
LL_LPTIM_ISR_ARRM	Autoreload match
LL_LPTIM_ISR_EXTTRIG	External trigger edge event
LL_LPTIM_ISR_CMPOK	Compare register update OK
LL_LPTIM_ISR_ARROK	Autoreload register update OK
LL_LPTIM_ISR_UP	Counter direction change down to up
LL_LPTIM_ISR_DOWN	Counter direction change up to down

IT Defines

LL_LPTIM_IER_CMPMIE	Compare match Interrupt Enable
LL_LPTIM_IER_ARRMIE	Autoreload match Interrupt Enable
LL_LPTIM_IER_EXTTRIGIE	External trigger valid edge Interrupt Enable
LL_LPTIM_IER_CMPOKIE	Compare register update OK Interrupt Enable
LL_LPTIM_IER_ARROKIE	Autoreload register update OK Interrupt Enable
LL_LPTIM_IER_UPIE	Direction change to UP Interrupt Enable
LL_LPTIM_IER_DOWNIE	Direction change to down Interrupt Enable

Operating Mode

LL_LPTIM_OPERATING_MODE_CONTINUOUS	LP Timer starts in continuous mode
LL_LPTIM_OPERATING_MODE_ONESHOT	LP Tilmer starts in single mode

Output Polarity

LL_LPTIM_OUTPUT_POLARITY_REGULAR	The LPTIM output reflects the compare results between LPTIMx_ARR and LPTIMx_CMP registers

LL_LPTIM_OUTPUT_POLARITY_INVERSE	The LPTIM output reflects the inverse of the compare results between LPTIMx_ARR and LPTIMx_CMP registers

Output Waveform Type

LL_LPTIM_OUTPUT_WAVEFORM_PWM	LPTIM generates either a PWM waveform or a One pulse waveform depending on chosen operating mode CONTINOUS or SINGLE
LL_LPTIM_OUTPUT_WAVEFORM_SETONCE	LPTIM generates a Set Once waveform

Prescaler Value

LL_LPTIM_PRESCALER_DIV1	Prescaler division factor is set to 1
LL_LPTIM_PRESCALER_DIV2	Prescaler division factor is set to 2
LL_LPTIM_PRESCALER_DIV4	Prescaler division factor is set to 4
LL_LPTIM_PRESCALER_DIV8	Prescaler division factor is set to 8
LL_LPTIM_PRESCALER_DIV16	Prescaler division factor is set to 16
LL_LPTIM_PRESCALER_DIV32	Prescaler division factor is set to 32
LL_LPTIM_PRESCALER_DIV64	Prescaler division factor is set to 64
LL_LPTIM_PRESCALER_DIV128	Prescaler division factor is set to 128

Trigger Filter

LL_LPTIM_TRIG_FILTER_NONE	Any trigger active level change is considered as a valid trigger
LL_LPTIM_TRIG_FILTER_2	Trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger
LL_LPTIM_TRIG_FILTER_4	Trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger
LL_LPTIM_TRIG_FILTER_8	Trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger

Trigger Polarity

LL_LPTIM_TRIG_POLARITY_RISING	LPTIM counter starts when a rising edge is detected
LL_LPTIM_TRIG_POLARITY_FALLING	LPTIM counter starts when a falling edge is detected
LL_LPTIM_TRIG_POLARITY_RISING_FALLING	LPTIM counter starts when a rising or a falling edge is detected

Trigger Source

LL_LPTIM_TRIG_SOURCE_GPIO	External input trigger is connected to TIMx_ETR input
LL_LPTIM_TRIG_SOURCE_RTCALARMA	External input trigger is connected to RTC Alarm A
LL_LPTIM_TRIG_SOURCE_RTCALARMB	External input trigger is connected to RTC Alarm B

LL_LPTIM_TRIG_SOURCE_RTCTAMP1	External input trigger is connected to RTC Tamper 1
LL_LPTIM_TRIG_SOURCE_RTCTAMP2	External input trigger is connected to RTC Tamper 2
LL_LPTIM_TRIG_SOURCE_RTCTAMP3	External input trigger is connected to RTC Tamper 3
LL_LPTIM_TRIG_SOURCE_COMP1	External input trigger is connected to COMP1 output
LL_LPTIM_TRIG_SOURCE_COMP2	External input trigger is connected to COMP2 output

Update Mode

LL_LPTIM_UPDATE_MODE_IMMEDIATE	Preload is disabled: registers are updated after each APB bus write access
LL_LPTIM_UPDATE_MODE_ENDOFPERIOD	preload is enabled: registers are updated at the end of the current LPTIM period

Common Write and read registers Macros

LL_LPTIM_WriteReg

Description:

- Write a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPTIM_ReadReg

Description:

- Read a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be read

Return value:

- Register: value

88 LL LPUART Generic Driver

88.1 LPUART Firmware driver registers structures

88.1.1 LL_LPUART_InitTypeDef

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***

Field Documentation

- ***uint32_t LL_LPUART_InitTypeDef::PrescalerValue***
  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***LPUART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_LPUART_SetPrescaler()**.
- ***uint32_t LL_LPUART_InitTypeDef::BaudRate***
  This field defines expected LPUART communication baud rate.This feature can be modified afterwards using unitary function **LL_LPUART_SetBaudRate()**.
- ***uint32_t LL_LPUART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***LPUART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_LPUART_SetDataWidth()**.
- ***uint32_t LL_LPUART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***LPUART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_LPUART_SetStopBitsLength()**.
- ***uint32_t LL_LPUART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***LPUART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_LPUART_SetParity()**.
- ***uint32_t LL_LPUART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_LPUART_SetTransferDirection()**.
- ***uint32_t LL_LPUART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_LPUART_SetHWFlowCtrl()**.

88.2 LPUART Firmware driver API description

88.2.1 Detailed description of functions

LL_LPUART_Enable

Function name	**__STATIC_INLINE void LL_LPUART_Enable (USART_TypeDef**

*** LPUARTx)**

Function description	LPUART Enable.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 UE LL_LPUART_Enable

LL_LPUART_Disable

Function name	**__STATIC_INLINE void LL_LPUART_Disable (USART_TypeDef * LPUARTx)**
Function description	LPUART Disable.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Notes	• When LPUART is disabled, LPUART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the LPUART is kept, but all the status flags, in the LPUARTx_ISR are set to their default values. • In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the LPUART_ISR to be set before resetting the UE bit. The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.
Reference Manual to LL API cross reference:	• CR1 UE LL_LPUART_Disable

LL_LPUART_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabled (USART_TypeDef * LPUARTx)**
Function description	Indicate if LPUART is enabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 UE LL_LPUART_IsEnabled

LL_LPUART_EnableFIFO

Function name	**__STATIC_INLINE void LL_LPUART_EnableFIFO (USART_TypeDef * LPUARTx)**
Function description	FIFO Mode Enable.

Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 FIFOEN LL_LPUART_EnableFIFO

LL_LPUART_DisableFIFO

Function name	**__STATIC_INLINE void LL_LPUART_DisableFIFO (USART_TypeDef * LPUARTx)**
Function description	FIFO Mode Disable.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 FIFOEN LL_LPUART_DisableFIFO

LL_LPUART_IsEnabledFIFO

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledFIFO (USART_TypeDef * LPUARTx)**
Function description	Indicate if FIFO Mode is enabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 FIFOEN LL_LPUART_IsEnabledFIFO

LL_LPUART_SetTXFIFOThreshold

Function name	**__STATIC_INLINE void LL_LPUART_SetTXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)**
Function description	Configure TX FIFO Threshold.
Parameters	• **LPUARTx:** LPUART Instance • **Threshold:** This parameter can be one of the following values: – LL_LPUART_FIFOTHRESHOLD_1_8 – LL_LPUART_FIFOTHRESHOLD_1_4 – LL_LPUART_FIFOTHRESHOLD_1_2 – LL_LPUART_FIFOTHRESHOLD_3_4 – LL_LPUART_FIFOTHRESHOLD_7_8 – LL_LPUART_FIFOTHRESHOLD_8_8
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 TXFTCFG LL_LPUART_SetTXFIFOThreshold

LL_LPUART_GetTXFIFOThreshold

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetTXFIFOThreshold (USART_TypeDef * LPUARTx)**
Function description	Return TX FIFO Threshold Configuration.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_FIFOTHRESHOLD_1_8 – LL_LPUART_FIFOTHRESHOLD_1_4 – LL_LPUART_FIFOTHRESHOLD_1_2 – LL_LPUART_FIFOTHRESHOLD_3_4 – LL_LPUART_FIFOTHRESHOLD_7_8 – LL_LPUART_FIFOTHRESHOLD_8_8
Reference Manual to LL API cross reference:	• CR3 TXFTCFG LL_LPUART_GetTXFIFOThreshold

LL_LPUART_SetRXFIFOThreshold

Function name	**__STATIC_INLINE void LL_LPUART_SetRXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)**
Function description	Configure RX FIFO Threshold.
Parameters	• **LPUARTx:** LPUART Instance • **Threshold:** This parameter can be one of the following values: – LL_LPUART_FIFOTHRESHOLD_1_8 – LL_LPUART_FIFOTHRESHOLD_1_4 – LL_LPUART_FIFOTHRESHOLD_1_2 – LL_LPUART_FIFOTHRESHOLD_3_4 – LL_LPUART_FIFOTHRESHOLD_7_8 – LL_LPUART_FIFOTHRESHOLD_8_8
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 RXFTCFG LL_LPUART_SetRXFIFOThreshold

LL_LPUART_GetRXFIFOThreshold

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetRXFIFOThreshold (USART_TypeDef * LPUARTx)**
Function description	Return RX FIFO Threshold Configuration.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_FIFOTHRESHOLD_1_8 – LL_LPUART_FIFOTHRESHOLD_1_4 – LL_LPUART_FIFOTHRESHOLD_1_2 – LL_LPUART_FIFOTHRESHOLD_3_4 – LL_LPUART_FIFOTHRESHOLD_7_8

	– LL_LPUART_FIFOTHRESHOLD_8_8
Reference Manual to LL API cross reference:	• CR3 RXFTCFG LL_LPUART_GetRXFIFOThreshold

LL_LPUART_ConfigFIFOsThreshold

Function name	**__STATIC_INLINE void LL_LPUART_ConfigFIFOsThreshold (USART_TypeDef * LPUARTx, uint32_t TXThreshold, uint32_t RXThreshold)**
Function description	Configure TX and RX FIFOs Threshold.
Parameters	• **LPUARTx:** LPUART Instance • **TXThreshold:** This parameter can be one of the following values: – LL_LPUART_FIFOTHRESHOLD_1_8 – LL_LPUART_FIFOTHRESHOLD_1_4 – LL_LPUART_FIFOTHRESHOLD_1_2 – LL_LPUART_FIFOTHRESHOLD_3_4 – LL_LPUART_FIFOTHRESHOLD_7_8 – LL_LPUART_FIFOTHRESHOLD_8_8 • **RXThreshold:** This parameter can be one of the following values: – LL_LPUART_FIFOTHRESHOLD_1_8 – LL_LPUART_FIFOTHRESHOLD_1_4 – LL_LPUART_FIFOTHRESHOLD_1_2 – LL_LPUART_FIFOTHRESHOLD_3_4 – LL_LPUART_FIFOTHRESHOLD_7_8 – LL_LPUART_FIFOTHRESHOLD_8_8
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 TXFTCFG LL_LPUART_ConfigFIFOsThreshold • CR3 RXFTCFG LL_LPUART_ConfigFIFOsThreshold

LL_LPUART_EnableInStopMode

Function name	**__STATIC_INLINE void LL_LPUART_EnableInStopMode (USART_TypeDef * LPUARTx)**
Function description	LPUART enabled in STOP Mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Notes	• When this function is enabled, LPUART is able to wake up the MCU from Stop mode, provided that LPUART clock selection is HSI or LSE in RCC.
Reference Manual to LL API cross reference:	• CR1 UESM LL_LPUART_EnableInStopMode

LL_LPUART_DisableInStopMode

Function name	**__STATIC_INLINE void LL_LPUART_DisableInStopMode (USART_TypeDef * LPUARTx)**
Function description	LPUART disabled in STOP Mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Notes	• When this function is disabled, LPUART is not able to wake up the MCU from Stop mode
Reference Manual to LL API cross reference:	• CR1 UESM LL_LPUART_DisableInStopMode

LL_LPUART_IsEnabledInStopMode

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledInStopMode (USART_TypeDef * LPUARTx)**
Function description	Indicate if LPUART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 UESM LL_LPUART_IsEnabledInStopMode

LL_LPUART_EnableDirectionRx

Function name	**__STATIC_INLINE void LL_LPUART_EnableDirectionRx (USART_TypeDef * LPUARTx)**
Function description	Receiver Enable (Receiver is enabled and begins searching for a start bit)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RE LL_LPUART_EnableDirectionRx

LL_LPUART_DisableDirectionRx

Function name	**__STATIC_INLINE void LL_LPUART_DisableDirectionRx (USART_TypeDef * LPUARTx)**
Function description	Receiver Disable.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• CR1 RE LL_LPUART_DisableDirectionRx

LL_LPUART_EnableDirectionTx

Function name	**__STATIC_INLINE void LL_LPUART_EnableDirectionTx (USART_TypeDef * LPUARTx)**
Function description	Transmitter Enable.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TE LL_LPUART_EnableDirectionTx

LL_LPUART_DisableDirectionTx

Function name	**__STATIC_INLINE void LL_LPUART_DisableDirectionTx (USART_TypeDef * LPUARTx)**
Function description	Transmitter Disable.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TE LL_LPUART_DisableDirectionTx

LL_LPUART_SetTransferDirection

Function name	**__STATIC_INLINE void LL_LPUART_SetTransferDirection (USART_TypeDef * LPUARTx, uint32_t TransferDirection)**
Function description	Configure simultaneously enabled/disabled states of Transmitter and Receiver.
Parameters	• **LPUARTx:** LPUART Instance • **TransferDirection:** This parameter can be one of the following values: – LL_LPUART_DIRECTION_NONE – LL_LPUART_DIRECTION_RX – LL_LPUART_DIRECTION_TX – LL_LPUART_DIRECTION_TX_RX
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RE LL_LPUART_SetTransferDirection • CR1 TE LL_LPUART_SetTransferDirection

LL_LPUART_GetTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetTransferDirection**

	(USART_TypeDef * LPUARTx)
Function description	Return enabled/disabled states of Transmitter and Receiver.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_DIRECTION_NONE – LL_LPUART_DIRECTION_RX – LL_LPUART_DIRECTION_TX – LL_LPUART_DIRECTION_TX_RX
Reference Manual to LL API cross reference:	• CR1 RE LL_LPUART_GetTransferDirection • CR1 TE LL_LPUART_GetTransferDirection

LL_LPUART_SetParity

Function name	**__STATIC_INLINE void LL_LPUART_SetParity (USART_TypeDef * LPUARTx, uint32_t Parity)**
Function description	Configure Parity (enabled/disabled and parity mode if enabled)
Parameters	• **LPUARTx:** LPUART Instance • **Parity:** This parameter can be one of the following values: – LL_LPUART_PARITY_NONE – LL_LPUART_PARITY_EVEN – LL_LPUART_PARITY_ODD
Return values	• **None:**
Notes	• This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (depending on data width) and parity is checked on the received data.
Reference Manual to LL API cross reference:	• CR1 PS LL_LPUART_SetParity • CR1 PCE LL_LPUART_SetParity

LL_LPUART_GetParity

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetParity (USART_TypeDef * LPUARTx)**
Function description	Return Parity configuration (enabled/disabled and parity mode if enabled)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_PARITY_NONE – LL_LPUART_PARITY_EVEN – LL_LPUART_PARITY_ODD
Reference Manual to LL API cross reference:	• CR1 PS LL_LPUART_GetParity • CR1 PCE LL_LPUART_GetParity

LL_LPUART_SetWakeUpMethod

Function name	**__STATIC_INLINE void LL_LPUART_SetWakeUpMethod (USART_TypeDef * LPUARTx, uint32_t Method)**
Function description	Set Receiver Wake Up method from Mute mode.
Parameters	• **LPUARTx:** LPUART Instance • **Method:** This parameter can be one of the following values: – LL_LPUART_WAKEUP_IDLELINE – LL_LPUART_WAKEUP_ADDRESSMARK
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 WAKE LL_LPUART_SetWakeUpMethod

LL_LPUART_GetWakeUpMethod

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetWakeUpMethod (USART_TypeDef * LPUARTx)**
Function description	Return Receiver Wake Up method from Mute mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_WAKEUP_IDLELINE – LL_LPUART_WAKEUP_ADDRESSMARK
Reference Manual to LL API cross reference:	• CR1 WAKE LL_LPUART_GetWakeUpMethod

LL_LPUART_SetDataWidth

Function name	**__STATIC_INLINE void LL_LPUART_SetDataWidth (USART_TypeDef * LPUARTx, uint32_t DataWidth)**
Function description	Set Word length (nb of data bits, excluding start and stop bits)
Parameters	• **LPUARTx:** LPUART Instance • **DataWidth:** This parameter can be one of the following values: – LL_LPUART_DATAWIDTH_7B – LL_LPUART_DATAWIDTH_8B – LL_LPUART_DATAWIDTH_9B
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 M LL_LPUART_SetDataWidth

LL_LPUART_GetDataWidth

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetDataWidth (USART_TypeDef * LPUARTx)**

Function description	Return Word length (i.e.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_DATAWIDTH_7B – LL_LPUART_DATAWIDTH_8B – LL_LPUART_DATAWIDTH_9B
Reference Manual to LL API cross reference:	• CR1 M LL_LPUART_GetDataWidth

LL_LPUART_EnableMuteMode

Function name	**__STATIC_INLINE void LL_LPUART_EnableMuteMode (USART_TypeDef * LPUARTx)**
Function description	Allow switch between Mute Mode and Active mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 MME LL_LPUART_EnableMuteMode

LL_LPUART_DisableMuteMode

Function name	**__STATIC_INLINE void LL_LPUART_DisableMuteMode (USART_TypeDef * LPUARTx)**
Function description	Prevent Mute Mode use.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 MME LL_LPUART_DisableMuteMode

LL_LPUART_IsEnabledMuteMode

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledMuteMode (USART_TypeDef * LPUARTx)**
Function description	Indicate if switch between Mute Mode and Active mode is allowed.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 MME LL_LPUART_IsEnabledMuteMode

LL_LPUART_SetPrescaler

Function name	**__STATIC_INLINE void LL_LPUART_SetPrescaler**

	(USART_TypeDef * LPUARTx, uint32_t PrescalerValue)
Function description	Configure Clock source prescaler for baudrate generator and oversampling.
Parameters	• **LPUARTx:** LPUART Instance • **PrescalerValue:** This parameter can be one of the following values: – LL_LPUART_PRESCALER_DIV1 – LL_LPUART_PRESCALER_DIV2 – LL_LPUART_PRESCALER_DIV4 – LL_LPUART_PRESCALER_DIV6 – LL_LPUART_PRESCALER_DIV8 – LL_LPUART_PRESCALER_DIV10 – LL_LPUART_PRESCALER_DIV12 – LL_LPUART_PRESCALER_DIV16 – LL_LPUART_PRESCALER_DIV32 – LL_LPUART_PRESCALER_DIV64 – LL_LPUART_PRESCALER_DIV128 – LL_LPUART_PRESCALER_DIV256
Return values	• **None:**
Reference Manual to LL API cross reference:	• PRESC PRESCALER LL_LPUART_SetPrescaler

LL_LPUART_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetPrescaler (USART_TypeDef * LPUARTx)**
Function description	Retrieve the Clock source prescaler for baudrate generator and oversampling.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_PRESCALER_DIV1 – LL_LPUART_PRESCALER_DIV2 – LL_LPUART_PRESCALER_DIV4 – LL_LPUART_PRESCALER_DIV6 – LL_LPUART_PRESCALER_DIV8 – LL_LPUART_PRESCALER_DIV10 – LL_LPUART_PRESCALER_DIV12 – LL_LPUART_PRESCALER_DIV16 – LL_LPUART_PRESCALER_DIV32 – LL_LPUART_PRESCALER_DIV64 – LL_LPUART_PRESCALER_DIV128 – LL_LPUART_PRESCALER_DIV256
Reference Manual to LL API cross reference:	• PRESC PRESCALER LL_LPUART_GetPrescaler

LL_LPUART_SetStopBitsLength

Function name	**__STATIC_INLINE void LL_LPUART_SetStopBitsLength (USART_TypeDef * LPUARTx, uint32_t StopBits)**
Function description	Set the length of the stop bits.
Parameters	• **LPUARTx:** LPUART Instance • **StopBits:** This parameter can be one of the following values: – LL_LPUART_STOPBITS_1 – LL_LPUART_STOPBITS_2
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 STOP LL_LPUART_SetStopBitsLength

LL_LPUART_GetStopBitsLength

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetStopBitsLength (USART_TypeDef * LPUARTx)**
Function description	Retrieve the length of the stop bits.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_STOPBITS_1 – LL_LPUART_STOPBITS_2
Reference Manual to LL API cross reference:	• CR2 STOP LL_LPUART_GetStopBitsLength

LL_LPUART_ConfigCharacter

Function name	**__STATIC_INLINE void LL_LPUART_ConfigCharacter (USART_TypeDef * LPUARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)**
Function description	Configure Character frame format (Datawidth, Parity control, Stop Bits)
Parameters	• **LPUARTx:** LPUART Instance • **DataWidth:** This parameter can be one of the following values: – LL_LPUART_DATAWIDTH_7B – LL_LPUART_DATAWIDTH_8B – LL_LPUART_DATAWIDTH_9B • **Parity:** This parameter can be one of the following values: – LL_LPUART_PARITY_NONE – LL_LPUART_PARITY_EVEN – LL_LPUART_PARITY_ODD • **StopBits:** This parameter can be one of the following values: – LL_LPUART_STOPBITS_1 – LL_LPUART_STOPBITS_2
Return values	• **None:**

Notes	• Call of this function is equivalent to following function call sequence: Data Width configuration using LL_LPUART_SetDataWidth() functionParity Control and mode configuration using LL_LPUART_SetParity() functionStop bits configuration using LL_LPUART_SetStopBitsLength() function
Reference Manual to LL API cross reference:	• CR1 PS LL_LPUART_ConfigCharacter • CR1 PCE LL_LPUART_ConfigCharacter • CR1 M LL_LPUART_ConfigCharacter • CR2 STOP LL_LPUART_ConfigCharacter

LL_LPUART_SetTXRXSwap

Function name	**__STATIC_INLINE void LL_LPUART_SetTXRXSwap (USART_TypeDef * LPUARTx, uint32_t SwapConfig)**
Function description	Configure TX/RX pins swapping setting.
Parameters	• **LPUARTx:** LPUART Instance • **SwapConfig:** This parameter can be one of the following values: – LL_LPUART_TXRX_STANDARD – LL_LPUART_TXRX_SWAPPED
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 SWAP LL_LPUART_SetTXRXSwap

LL_LPUART_GetTXRXSwap

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetTXRXSwap (USART_TypeDef * LPUARTx)**
Function description	Retrieve TX/RX pins swapping configuration.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_TXRX_STANDARD – LL_LPUART_TXRX_SWAPPED
Reference Manual to LL API cross reference:	• CR2 SWAP LL_LPUART_GetTXRXSwap

LL_LPUART_SetRXPinLevel

Function name	**__STATIC_INLINE void LL_LPUART_SetRXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)**
Function description	Configure RX pin active level logic.
Parameters	• **LPUARTx:** LPUART Instance • **PinInvMethod:** This parameter can be one of the following values: – LL_LPUART_RXPIN_LEVEL_STANDARD

	– LL_LPUART_RXPIN_LEVEL_INVERTED
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXINV LL_LPUART_SetRXPinLevel

LL_LPUART_GetRXPinLevel

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetRXPinLevel (USART_TypeDef * LPUARTx)**
Function description	Retrieve RX pin active level logic configuration.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_RXPIN_LEVEL_STANDARD – LL_LPUART_RXPIN_LEVEL_INVERTED
Reference Manual to LL API cross reference:	• CR2 RXINV LL_LPUART_GetRXPinLevel

LL_LPUART_SetTXPinLevel

Function name	**__STATIC_INLINE void LL_LPUART_SetTXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)**
Function description	Configure TX pin active level logic.
Parameters	• **LPUARTx:** LPUART Instance • **PinInvMethod:** This parameter can be one of the following values: – LL_LPUART_TXPIN_LEVEL_STANDARD – LL_LPUART_TXPIN_LEVEL_INVERTED
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXINV LL_LPUART_SetTXPinLevel

LL_LPUART_GetTXPinLevel

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetTXPinLevel (USART_TypeDef * LPUARTx)**
Function description	Retrieve TX pin active level logic configuration.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_TXPIN_LEVEL_STANDARD – LL_LPUART_TXPIN_LEVEL_INVERTED
Reference Manual to LL API cross reference:	• CR2 TXINV LL_LPUART_GetTXPinLevel

LL_LPUART_SetBinaryDataLogic

Function name	**__STATIC_INLINE void LL_LPUART_SetBinaryDataLogic (USART_TypeDef * LPUARTx, uint32_t DataLogic)**
Function description	Configure Binary data logic.
Parameters	• **LPUARTx:** LPUART Instance • **DataLogic:** This parameter can be one of the following values: – LL_LPUART_BINARY_LOGIC_POSITIVE – LL_LPUART_BINARY_LOGIC_NEGATIVE
Return values	• **None:**
Notes	• Allow to define how Logical data from the data register are send/received: either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)
Reference Manual to LL API cross reference:	• CR2 DATAINV LL_LPUART_SetBinaryDataLogic

LL_LPUART_GetBinaryDataLogic

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetBinaryDataLogic (USART_TypeDef * LPUARTx)**
Function description	Retrieve Binary data configuration.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_BINARY_LOGIC_POSITIVE – LL_LPUART_BINARY_LOGIC_NEGATIVE
Reference Manual to LL API cross reference:	• CR2 DATAINV LL_LPUART_GetBinaryDataLogic

LL_LPUART_SetTransferBitOrder

Function name	**__STATIC_INLINE void LL_LPUART_SetTransferBitOrder (USART_TypeDef * LPUARTx, uint32_t BitOrder)**
Function description	Configure transfer bit order (either Less or Most Significant Bit First)
Parameters	• **LPUARTx:** LPUART Instance • **BitOrder:** This parameter can be one of the following values: – LL_LPUART_BITORDER_LSBFIRST – LL_LPUART_BITORDER_MSBFIRST
Return values	• **None:**
Notes	• MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.
Reference Manual to LL API cross	• CR2 MSBFIRST LL_LPUART_SetTransferBitOrder

reference:

LL_LPUART_GetTransferBitOrder

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetTransferBitOrder (USART_TypeDef * LPUARTx)**
Function description	Return transfer bit order (either Less or Most Significant Bit First)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_BITORDER_LSBFIRST – LL_LPUART_BITORDER_MSBFIRST
Notes	• MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.
Reference Manual to LL API cross reference:	• CR2 MSBFIRST LL_LPUART_GetTransferBitOrder

LL_LPUART_ConfigNodeAddress

Function name	**__STATIC_INLINE void LL_LPUART_ConfigNodeAddress (USART_TypeDef * LPUARTx, uint32_t AddressLen, uint32_t NodeAddress)**
Function description	Set Address of the LPUART node.
Parameters	• **LPUARTx:** LPUART Instance • **AddressLen:** This parameter can be one of the following values: – LL_LPUART_ADDRESS_DETECT_4B – LL_LPUART_ADDRESS_DETECT_7B • **NodeAddress:** 4 or 7 bit Address of the LPUART node.
Return values	• **None:**
Notes	• This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection. • 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)
Reference Manual to LL API cross reference:	• CR2 ADD LL_LPUART_ConfigNodeAddress • CR2 ADDM7 LL_LPUART_ConfigNodeAddress

LL_LPUART_GetNodeAddress

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddress (USART_TypeDef * LPUARTx)**
Function description	Return 8 bit Address of the LPUART node as set in ADD field of CR2.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Address:** of the LPUART node (Value between Min_Data=0 and Max_Data=255)
Notes	• If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)
Reference Manual to LL API cross reference:	• CR2 ADD LL_LPUART_GetNodeAddress

LL_LPUART_GetNodeAddressLen

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddressLen (USART_TypeDef * LPUARTx)**
Function description	Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_ADDRESS_DETECT_4B – LL_LPUART_ADDRESS_DETECT_7B
Reference Manual to LL API cross reference:	• CR2 ADDM7 LL_LPUART_GetNodeAddressLen

LL_LPUART_EnableRTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_LPUART_EnableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)**
Function description	Enable RTS HW Flow Control.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 RTSE LL_LPUART_EnableRTSHWFlowCtrl

LL_LPUART_DisableRTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_LPUART_DisableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)**

Function description	Disable RTS HW Flow Control.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 RTSE LL_LPUART_DisableRTSHWFlowCtrl

LL_LPUART_EnableCTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_LPUART_EnableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)**
Function description	Enable CTS HW Flow Control.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 CTSE LL_LPUART_EnableCTSHWFlowCtrl

LL_LPUART_DisableCTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_LPUART_DisableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)**
Function description	Disable CTS HW Flow Control.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 CTSE LL_LPUART_DisableCTSHWFlowCtrl

LL_LPUART_SetHWFlowCtrl

Function name	**__STATIC_INLINE void LL_LPUART_SetHWFlowCtrl (USART_TypeDef * LPUARTx, uint32_t HardwareFlowControl)**
Function description	Configure HW Flow Control mode (both CTS and RTS)
Parameters	• **LPUARTx:** LPUART Instance • **HardwareFlowControl:** This parameter can be one of the following values: – LL_LPUART_HWCONTROL_NONE – LL_LPUART_HWCONTROL_RTS – LL_LPUART_HWCONTROL_CTS – LL_LPUART_HWCONTROL_RTS_CTS
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 RTSE LL_LPUART_SetHWFlowCtrl • CR3 CTSE LL_LPUART_SetHWFlowCtrl

LL_LPUART_GetHWFlowCtrl

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetHWFlowCtrl (USART_TypeDef * LPUARTx)**
Function description	Return HW Flow Control configuration (both CTS and RTS)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_HWCONTROL_NONE – LL_LPUART_HWCONTROL_RTS – LL_LPUART_HWCONTROL_CTS – LL_LPUART_HWCONTROL_RTS_CTS
Reference Manual to LL API cross reference:	• CR3 RTSE LL_LPUART_GetHWFlowCtrl • CR3 CTSE LL_LPUART_GetHWFlowCtrl

LL_LPUART_EnableOverrunDetect

Function name	**__STATIC_INLINE void LL_LPUART_EnableOverrunDetect (USART_TypeDef * LPUARTx)**
Function description	Enable Overrun detection.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 OVRDIS LL_LPUART_EnableOverrunDetect

LL_LPUART_DisableOverrunDetect

Function name	**__STATIC_INLINE void LL_LPUART_DisableOverrunDetect (USART_TypeDef * LPUARTx)**
Function description	Disable Overrun detection.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 OVRDIS LL_LPUART_DisableOverrunDetect

LL_LPUART_IsEnabledOverrunDetect

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledOverrunDetect (USART_TypeDef * LPUARTx)**
Function description	Indicate if Overrun detection is enabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to	• CR3 OVRDIS LL_LPUART_IsEnabledOverrunDetect

LL API cross reference:	

LL_LPUART_SetWKUPType

Function name	**__STATIC_INLINE void LL_LPUART_SetWKUPType (USART_TypeDef * LPUARTx, uint32_t Type)**
Function description	Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	• **LPUARTx:** LPUART Instance • **Type:** This parameter can be one of the following values: – LL_LPUART_WAKEUP_ON_ADDRESS – LL_LPUART_WAKEUP_ON_STARTBIT – LL_LPUART_WAKEUP_ON_RXNE
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 WUS LL_LPUART_SetWKUPType

LL_LPUART_GetWKUPType

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetWKUPType (USART_TypeDef * LPUARTx)**
Function description	Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_WAKEUP_ON_ADDRESS – LL_LPUART_WAKEUP_ON_STARTBIT – LL_LPUART_WAKEUP_ON_RXNE
Reference Manual to LL API cross reference:	• CR3 WUS LL_LPUART_GetWKUPType

LL_LPUART_SetBaudRate

Function name	**__STATIC_INLINE void LL_LPUART_SetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t BaudRate)**
Function description	Configure LPUART BRR register for achieving expected Baud Rate value.
Parameters	• **LPUARTx:** LPUART Instance • **PeriphClk:** Peripheral Clock • **BaudRate:** Baud Rate
Return values	• **None:**
Notes	• Compute and set LPUARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock and expected Baud Rate values • Peripheral clock and Baud Rate values provided as function parameters should be valid (Baud rate value != 0).

	• Provided that LPUARTx_BRR must be > = 0x300 and LPUART_BRR is 20-bit, a care should be taken when generating high baud rates using high PeriphClk values. PeriphClk must be in the range [3 x BaudRate, 4096 x BaudRate].
Reference Manual to LL API cross reference:	• BRR BRR LL_LPUART_SetBaudRate

LL_LPUART_GetBaudRate

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue)**
Function description	Return current Baud Rate value, according to LPUARTDIV present in BRR register (full BRR content), and to used Peripheral Clock values.
Parameters	• **LPUARTx:** LPUART Instance • **PeriphClk:** Peripheral Clock
Return values	• **Baud:** Rate
Notes	• In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.
Reference Manual to LL API cross reference:	• BRR BRR LL_LPUART_GetBaudRate

LL_LPUART_EnableHalfDuplex

Function name	**__STATIC_INLINE void LL_LPUART_EnableHalfDuplex (USART_TypeDef * LPUARTx)**
Function description	Enable Single Wire Half-Duplex mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_LPUART_EnableHalfDuplex

LL_LPUART_DisableHalfDuplex

Function name	**__STATIC_INLINE void LL_LPUART_DisableHalfDuplex (USART_TypeDef * LPUARTx)**
Function description	Disable Single Wire Half-Duplex mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_LPUART_DisableHalfDuplex

LL_LPUART_IsEnabledHalfDuplex

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledHalfDuplex (USART_TypeDef * LPUARTx)**
Function description	Indicate if Single Wire Half-Duplex mode is enabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_LPUART_IsEnabledHalfDuplex

LL_LPUART_SetDEDeassertionTime

Function name	**__STATIC_INLINE void LL_LPUART_SetDEDeassertionTime (USART_TypeDef * LPUARTx, uint32_t Time)**
Function description	Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	• **LPUARTx:** LPUART Instance • **Time:** Value between Min_Data=0 and Max_Data=31
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 DEDT LL_LPUART_SetDEDeassertionTime

LL_LPUART_GetDEDeassertionTime

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetDEDeassertionTime (USART_TypeDef * LPUARTx)**
Function description	Return DEDT (Driver Enable De-Assertion Time)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Time:** value expressed on 5 bits ([4:0] bits): c
Reference Manual to LL API cross reference:	• CR1 DEDT LL_LPUART_GetDEDeassertionTime

LL_LPUART_SetDEAssertionTime

Function name	**__STATIC_INLINE void LL_LPUART_SetDEAssertionTime (USART_TypeDef * LPUARTx, uint32_t Time)**
Function description	Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	• **LPUARTx:** LPUART Instance • **Time:** Value between Min_Data=0 and Max_Data=31
Return values	• **None:**
Reference Manual to	• CR1 DEAT LL_LPUART_SetDEAssertionTime

LL API cross reference:

LL_LPUART_GetDEAssertionTime

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetDEAssertionTime (USART_TypeDef * LPUARTx)**
Function description	Return DEAT (Driver Enable Assertion Time)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Time:** value expressed on 5 bits ([4:0] bits): Time Value between Min_Data=0 and Max_Data=31
Reference Manual to LL API cross reference:	• CR1 DEAT LL_LPUART_GetDEAssertionTime

LL_LPUART_EnableDEMode

Function name	**__STATIC_INLINE void LL_LPUART_EnableDEMode (USART_TypeDef * LPUARTx)**
Function description	Enable Driver Enable (DE) Mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DEM LL_LPUART_EnableDEMode

LL_LPUART_DisableDEMode

Function name	**__STATIC_INLINE void LL_LPUART_DisableDEMode (USART_TypeDef * LPUARTx)**
Function description	Disable Driver Enable (DE) Mode.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DEM LL_LPUART_DisableDEMode

LL_LPUART_IsEnabledDEMode

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDEMode (USART_TypeDef * LPUARTx)**
Function description	Indicate if Driver Enable (DE) Mode is enabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CR3 DEM LL_LPUART_IsEnabledDEMode

reference:

LL_LPUART_SetDESignalPolarity

Function name	**__STATIC_INLINE void LL_LPUART_SetDESignalPolarity (USART_TypeDef * LPUARTx, uint32_t Polarity)**
Function description	Select Driver Enable Polarity.
Parameters	• **LPUARTx:** LPUART Instance • **Polarity:** This parameter can be one of the following values: – LL_LPUART_DE_POLARITY_HIGH – LL_LPUART_DE_POLARITY_LOW
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DEP LL_LPUART_SetDESignalPolarity

LL_LPUART_GetDESignalPolarity

Function name	**__STATIC_INLINE uint32_t LL_LPUART_GetDESignalPolarity (USART_TypeDef * LPUARTx)**
Function description	Return Driver Enable Polarity.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Returned:** value can be one of the following values: – LL_LPUART_DE_POLARITY_HIGH – LL_LPUART_DE_POLARITY_LOW
Reference Manual to LL API cross reference:	• CR3 DEP LL_LPUART_GetDESignalPolarity

LL_LPUART_IsActiveFlag_PE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_PE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Parity Error Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR PE LL_LPUART_IsActiveFlag_PE

LL_LPUART_IsActiveFlag_FE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_FE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Framing Error Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR FE LL_LPUART_IsActiveFlag_FE

LL_LPUART_IsActiveFlag_NE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_NE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Noise error detected Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR NE LL_LPUART_IsActiveFlag_NE

LL_LPUART_IsActiveFlag_ORE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_ORE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART OverRun Error Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ORE LL_LPUART_IsActiveFlag_ORE

LL_LPUART_IsActiveFlag_IDLE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_IDLE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART IDLE line detected Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR IDLE LL_LPUART_IsActiveFlag_IDLE

LL_LPUART_IsActiveFlag_RXNE_RXFNE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXNE_RXFNE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Read Data Register or LPUART RX FIFO Not Empty Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RXNE_RXFNE LL_LPUART_IsActiveFlag_RXNE_RXFNE

LL_LPUART_IsActiveFlag_TC

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TC (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Transmission Complete Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TC LL_LPUART_IsActiveFlag_TC

LL_LPUART_IsActiveFlag_TXE_TXFNF

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXE_TXFNF (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Transmit Data Register Empty or LPUART TX FIFO Not Full Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TXE_TXFNF LL_LPUART_IsActiveFlag_TXE_TXFNF

LL_LPUART_IsActiveFlag_nCTS

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_nCTS (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART CTS interrupt Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CTSIF LL_LPUART_IsActiveFlag_nCTS

LL_LPUART_IsActiveFlag_CTS

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CTS (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART CTS Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CTS LL_LPUART_IsActiveFlag_CTS

LL_LPUART_IsActiveFlag_BUSY

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_BUSY (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Busy Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR BUSY LL_LPUART_IsActiveFlag_BUSY

LL_LPUART_IsActiveFlag_CM

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CM (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Character Match Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CMF LL_LPUART_IsActiveFlag_CM

LL_LPUART_IsActiveFlag_SBK

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_SBK (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Send Break Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SBKF LL_LPUART_IsActiveFlag_SBK

LL_LPUART_IsActiveFlag_RWU

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RWU (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Receive Wake Up from mute mode Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RWU LL_LPUART_IsActiveFlag_RWU

LL_LPUART_IsActiveFlag_WKUP

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_WKUP (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Wake Up from stop mode Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR WUF LL_LPUART_IsActiveFlag_WKUP

LL_LPUART_IsActiveFlag_TEACK

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TEACK (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Transmit Enable Acknowledge Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEACK LL_LPUART_IsActiveFlag_TEACK

LL_LPUART_IsActiveFlag_REACK

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_REACK (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Receive Enable Acknowledge Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR REACK LL_LPUART_IsActiveFlag_REACK

LL_LPUART_IsActiveFlag_TXFE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART TX FIFO Empty Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TXFE LL_LPUART_IsActiveFlag_TXFE

LL_LPUART_IsActiveFlag_RXFF

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFF (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART RX FIFO Full Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RXFF LL_LPUART_IsActiveFlag_RXFF

LL_LPUART_IsActiveFlag_TXFT

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFT (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART TX FIFO Threshold Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TXFT LL_LPUART_IsActiveFlag_TXFT

LL_LPUART_IsActiveFlag_RXFT

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFT (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART RX FIFO Threshold Flag is set or not.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RXFT LL_LPUART_IsActiveFlag_RXFT

LL_LPUART_ClearFlag_PE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_PE (USART_TypeDef * LPUARTx)**
Function description	Clear Parity Error Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• ICR PECF LL_LPUART_ClearFlag_PE

LL_LPUART_ClearFlag_FE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_FE (USART_TypeDef * LPUARTx)**
Function description	Clear Framing Error Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR FECF LL_LPUART_ClearFlag_FE

LL_LPUART_ClearFlag_NE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_NE (USART_TypeDef * LPUARTx)**
Function description	Clear Noise detected Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR NCF LL_LPUART_ClearFlag_NE

LL_LPUART_ClearFlag_ORE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_ORE (USART_TypeDef * LPUARTx)**
Function description	Clear OverRun Error Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR ORECF LL_LPUART_ClearFlag_ORE

LL_LPUART_ClearFlag_IDLE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_IDLE (USART_TypeDef * LPUARTx)**
Function description	Clear IDLE line detected Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross	• ICR IDLECF LL_LPUART_ClearFlag_IDLE

reference:

LL_LPUART_ClearFlag_TXFE

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_TXFE (USART_TypeDef * LPUARTx)**
Function description	Clear TX FIFO Empty Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR TXFECF LL_LPUART_ClearFlag_TXFE

LL_LPUART_ClearFlag_TC

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_TC (USART_TypeDef * LPUARTx)**
Function description	Clear Transmission Complete Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR TCCF LL_LPUART_ClearFlag_TC

LL_LPUART_ClearFlag_nCTS

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_nCTS (USART_TypeDef * LPUARTx)**
Function description	Clear CTS Interrupt Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR CTSCF LL_LPUART_ClearFlag_nCTS

LL_LPUART_ClearFlag_CM

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_CM (USART_TypeDef * LPUARTx)**
Function description	Clear Character Match Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR CMCF LL_LPUART_ClearFlag_CM

LL_LPUART_ClearFlag_WKUP

Function name	**__STATIC_INLINE void LL_LPUART_ClearFlag_WKUP (USART_TypeDef * LPUARTx)**
Function description	Clear Wake Up from stop mode Flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR WUCF LL_LPUART_ClearFlag_WKUP

LL_LPUART_EnableIT_IDLE

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_IDLE (USART_TypeDef * LPUARTx)**
Function description	Enable IDLE Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_LPUART_EnableIT_IDLE

LL_LPUART_EnableIT_RXNE_RXFNE

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)**
Function description	Enable RX Not Empty and RX FIFO Not Empty Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RXNEIE_RXFNEIE LL_LPUART_EnableIT_RXNE_RXFNE

LL_LPUART_EnableIT_TC

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_TC (USART_TypeDef * LPUARTx)**
Function description	Enable Transmission Complete Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TCIE LL_LPUART_EnableIT_TC

LL_LPUART_EnableIT_TXE_TXFNF

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)**
Function description	Enable TX Empty and TX FIFO Not Full Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TXEIE_TXFNFIE LL_LPUART_EnableIT_TXE_TXFNF

LL_LPUART_EnableIT_PE

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_PE (USART_TypeDef * LPUARTx)**
Function description	Enable Parity Error Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 PEIE LL_LPUART_EnableIT_PE

LL_LPUART_EnableIT_CM

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_CM (USART_TypeDef * LPUARTx)**
Function description	Enable Character Match Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 CMIE LL_LPUART_EnableIT_CM

LL_LPUART_EnableIT_TXFE

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_TXFE (USART_TypeDef * LPUARTx)**
Function description	Enable TX FIFO Empty Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TXFEIE LL_LPUART_EnableIT_TXFE

LL_LPUART_EnableIT_RXFF

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_RXFF (USART_TypeDef * LPUARTx)**
Function description	Enable RX FIFO Full Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RXFFIE LL_LPUART_EnableIT_RXFF

LL_LPUART_EnableIT_ERROR

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_ERROR (USART_TypeDef * LPUARTx)**
Function description	Enable Error Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Notes	• When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.
Reference Manual to LL API cross reference:	• CR3 EIE LL_LPUART_EnableIT_ERROR

LL_LPUART_EnableIT_CTS

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_CTS (USART_TypeDef * LPUARTx)**
Function description	Enable CTS Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 CTSIE LL_LPUART_EnableIT_CTS

LL_LPUART_EnableIT_WKUP

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_WKUP (USART_TypeDef * LPUARTx)**
Function description	Enable Wake Up from Stop Mode Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• CR3 WUFIE LL_LPUART_EnableIT_WKUP

LL_LPUART_EnableIT_TXFT

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_TXFT (USART_TypeDef * LPUARTx)**
Function description	Enable TX FIFO Threshold Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 TXFTIE LL_LPUART_EnableIT_TXFT

LL_LPUART_EnableIT_RXFT

Function name	**__STATIC_INLINE void LL_LPUART_EnableIT_RXFT (USART_TypeDef * LPUARTx)**
Function description	Enable RX FIFO Threshold Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 RXFTIE LL_LPUART_EnableIT_RXFT

LL_LPUART_DisableIT_IDLE

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_IDLE (USART_TypeDef * LPUARTx)**
Function description	Disable IDLE Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_LPUART_DisableIT_IDLE

LL_LPUART_DisableIT_RXNE_RXFNE

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)**
Function description	Disable RX Not Empty and RX FIFO Not Empty Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross	• CR1 RXNEIE_RXFNEIE

reference:	LL_LPUART_DisableIT_RXNE_RXFNE

LL_LPUART_DisableIT_TC

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_TC (USART_TypeDef * LPUARTx)**
Function description	Disable Transmission Complete Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TCIE LL_LPUART_DisableIT_TC

LL_LPUART_DisableIT_TXE_TXFNF

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)**
Function description	Disable TX Empty and TX FIFO Not Full Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TXEIE_TXFNFIE LL_LPUART_DisableIT_TXE_TXFNF

LL_LPUART_DisableIT_PE

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_PE (USART_TypeDef * LPUARTx)**
Function description	Disable Parity Error Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 PEIE LL_LPUART_DisableIT_PE

LL_LPUART_DisableIT_CM

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_CM (USART_TypeDef * LPUARTx)**
Function description	Disable Character Match Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 CMIE LL_LPUART_DisableIT_CM

LL_LPUART_DisableIT_TXFE

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_TXFE (USART_TypeDef * LPUARTx)**
Function description	Disable TX FIFO Empty Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TXFEIE LL_LPUART_DisableIT_TXFE

LL_LPUART_DisableIT_RXFF

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_RXFF (USART_TypeDef * LPUARTx)**
Function description	Disable RX FIFO Full Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RXFFIE LL_LPUART_DisableIT_RXFF

LL_LPUART_DisableIT_ERROR

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_ERROR (USART_TypeDef * LPUARTx)**
Function description	Disable Error Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Notes	• When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.
Reference Manual to LL API cross reference:	• CR3 EIE LL_LPUART_DisableIT_ERROR

LL_LPUART_DisableIT_CTS

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_CTS (USART_TypeDef * LPUARTx)**
Function description	Disable CTS Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• CR3 CTSIE LL_LPUART_DisableIT_CTS

LL_LPUART_DisableIT_WKUP

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_WKUP (USART_TypeDef * LPUARTx)**
Function description	Disable Wake Up from Stop Mode Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 WUFIE LL_LPUART_DisableIT_WKUP

LL_LPUART_DisableIT_TXFT

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_TXFT (USART_TypeDef * LPUARTx)**
Function description	Disable TX FIFO Threshold Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 TXFTIE LL_LPUART_DisableIT_TXFT

LL_LPUART_DisableIT_RXFT

Function name	**__STATIC_INLINE void LL_LPUART_DisableIT_RXFT (USART_TypeDef * LPUARTx)**
Function description	Disable RX FIFO Threshold Interrupt.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 RXFTIE LL_LPUART_DisableIT_RXFT

LL_LPUART_IsEnabledIT_IDLE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_IDLE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART IDLE Interrupt source is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CR1 IDLEIE LL_LPUART_IsEnabledIT_IDLE

reference:

LL_LPUART_IsEnabledIT_RXNE_RXFNE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART RX Not Empty and LPUART RX FIFO Not Empty Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RXNEIE_RXFNEIE LL_LPUART_IsEnabledIT_RXNE_RXFNE

LL_LPUART_IsEnabledIT_TC

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TC (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Transmission Complete Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TCIE LL_LPUART_IsEnabledIT_TC

LL_LPUART_IsEnabledIT_TXE_TXFNF

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXE_TXFNF (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART TX Empty and LPUART TX FIFO Not Full Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TXEIE_TXFNFIE LL_LPUART_IsEnabledIT_TXE_TXFNF

LL_LPUART_IsEnabledIT_PE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_PE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Parity Error Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 PEIE LL_LPUART_IsEnabledIT_PE

LL_LPUART_IsEnabledIT_CM

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CM (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Character Match Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 CMIE LL_LPUART_IsEnabledIT_CM

LL_LPUART_IsEnabledIT_TXFE

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFE (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART TX FIFO Empty Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TXFEIE LL_LPUART_IsEnabledIT_TXFE

LL_LPUART_IsEnabledIT_RXFF

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFF (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART RX FIFO Full Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RXFFIE LL_LPUART_IsEnabledIT_RXFF

LL_LPUART_IsEnabledIT_ERROR

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_ERROR (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Error Interrupt is enabled or disabled.

Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 EIE LL_LPUART_IsEnabledIT_ERROR

LL_LPUART_IsEnabledIT_CTS

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CTS (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART CTS Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 CTSIE LL_LPUART_IsEnabledIT_CTS

LL_LPUART_IsEnabledIT_WKUP

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_WKUP (USART_TypeDef * LPUARTx)**
Function description	Check if the LPUART Wake Up from Stop Mode Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 WUFIE LL_LPUART_IsEnabledIT_WKUP

LL_LPUART_IsEnabledIT_TXFT

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFT (USART_TypeDef * LPUARTx)**
Function description	Check if LPUART TX FIFO Threshold Interrupt is enabled or disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 TXFTIE LL_LPUART_IsEnabledIT_TXFT

LL_LPUART_IsEnabledIT_RXFT

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFT (USART_TypeDef * LPUARTx)**
Function description	Check if LPUART RX FIFO Threshold Interrupt is enabled or

	disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 RXFTIE LL_LPUART_IsEnabledIT_RXFT

LL_LPUART_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_LPUART_EnableDMAReq_RX (USART_TypeDef * LPUARTx)**
Function description	Enable DMA Mode for reception.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAR LL_LPUART_EnableDMAReq_RX

LL_LPUART_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_LPUART_DisableDMAReq_RX (USART_TypeDef * LPUARTx)**
Function description	Disable DMA Mode for reception.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAR LL_LPUART_DisableDMAReq_RX

LL_LPUART_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_RX (USART_TypeDef * LPUARTx)**
Function description	Check if DMA Mode is enabled for reception.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DMAR LL_LPUART_IsEnabledDMAReq_RX

LL_LPUART_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_LPUART_EnableDMAReq_TX (USART_TypeDef * LPUARTx)**

Function description	Enable DMA Mode for transmission.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAT LL_LPUART_EnableDMAReq_TX

LL_LPUART_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_LPUART_DisableDMAReq_TX (USART_TypeDef * LPUARTx)**
Function description	Disable DMA Mode for transmission.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAT LL_LPUART_DisableDMAReq_TX

LL_LPUART_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_TX (USART_TypeDef * LPUARTx)**
Function description	Check if DMA Mode is enabled for transmission.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DMAT LL_LPUART_IsEnabledDMAReq_TX

LL_LPUART_EnableDMADeactOnRxErr

Function name	**__STATIC_INLINE void LL_LPUART_EnableDMADeactOnRxErr (USART_TypeDef * LPUARTx)**
Function description	Enable DMA Disabling on Reception Error.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DDRE LL_LPUART_EnableDMADeactOnRxErr

LL_LPUART_DisableDMADeactOnRxErr

Function name	**__STATIC_INLINE void LL_LPUART_DisableDMADeactOnRxErr (USART_TypeDef ***

LPUARTx)

Function description	Disable DMA Disabling on Reception Error.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DDRE LL_LPUART_DisableDMADeactOnRxErr

LL_LPUART_IsEnabledDMADeactOnRxErr

Function name	**__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMADeactOnRxErr (USART_TypeDef * LPUARTx)**
Function description	Indicate if DMA Disabling on Reception Error is disabled.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DDRE LL_LPUART_IsEnabledDMADeactOnRxErr

LL_LPUART_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_LPUART_DMA_GetRegAddr (USART_TypeDef * LPUARTx, uint32_t Direction)**
Function description	Get the LPUART data register address used for DMA transfer.
Parameters	• **LPUARTx:** LPUART Instance • **Direction:** This parameter can be one of the following values: – LL_LPUART_DMA_REG_DATA_TRANSMIT – LL_LPUART_DMA_REG_DATA_RECEIVE
Return values	• **Address:** of data register
Reference Manual to LL API cross reference:	• RDR RDR LL_LPUART_DMA_GetRegAddr • • TDR TDR LL_LPUART_DMA_GetRegAddr

LL_LPUART_ReceiveData8

Function name	**__STATIC_INLINE uint8_t LL_LPUART_ReceiveData8 (USART_TypeDef * LPUARTx)**
Function description	Read Receiver Data register (Receive Data value, 8 bits)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Time:** Value between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• RDR RDR LL_LPUART_ReceiveData8

LL_LPUART_ReceiveData9

Function name	**__STATIC_INLINE uint16_t LL_LPUART_ReceiveData9 (USART_TypeDef * LPUARTx)**
Function description	Read Receiver Data register (Receive Data value, 9 bits)
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **Time:** Value between Min_Data=0x00 and Max_Data=0x1FF
Reference Manual to LL API cross reference:	• RDR RDR LL_LPUART_ReceiveData9

LL_LPUART_TransmitData8

Function name	**__STATIC_INLINE void LL_LPUART_TransmitData8 (USART_TypeDef * LPUARTx, uint8_t Value)**
Function description	Write in Transmitter Data Register (Transmit Data value, 8 bits)
Parameters	• **LPUARTx:** LPUART Instance • **Value:** between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• TDR TDR LL_LPUART_TransmitData8

LL_LPUART_TransmitData9

Function name	**__STATIC_INLINE void LL_LPUART_TransmitData9 (USART_TypeDef * LPUARTx, uint16_t Value)**
Function description	Write in Transmitter Data Register (Transmit Data value, 9 bits)
Parameters	• **LPUARTx:** LPUART Instance • **Value:** between Min_Data=0x00 and Max_Data=0x1FF
Return values	• **None:**
Reference Manual to LL API cross reference:	• TDR TDR LL_LPUART_TransmitData9

LL_LPUART_RequestBreakSending

Function name	**__STATIC_INLINE void LL_LPUART_RequestBreakSending (USART_TypeDef * LPUARTx)**
Function description	Request Break sending.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• RQR SBKRQ LL_LPUART_RequestBreakSending

LL_LPUART_RequestEnterMuteMode

Function name	**__STATIC_INLINE void LL_LPUART_RequestEnterMuteMode (USART_TypeDef * LPUARTx)**
Function description	Put LPUART in mute mode and set the RWU flag.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• RQR MMRQ LL_LPUART_RequestEnterMuteMode

LL_LPUART_RequestRxDataFlush

Function name	**__STATIC_INLINE void LL_LPUART_RequestRxDataFlush (USART_TypeDef * LPUARTx)**
Function description	Request a Receive Data flush.
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• RQR RXFRQ LL_LPUART_RequestRxDataFlush

LL_LPUART_DeInit

Function name	**ErrorStatus LL_LPUART_DeInit (USART_TypeDef * LPUARTx)**
Function description	De-initialize LPUART registers (Registers restored to their default values).
Parameters	• **LPUARTx:** LPUART Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: LPUART registers are de-initialized – ERROR: not applicable

LL_LPUART_Init

Function name	**ErrorStatus LL_LPUART_Init (USART_TypeDef * LPUARTx, LL_LPUART_InitTypeDef * LPUART_InitStruct)**
Function description	Initialize LPUART registers according to the specified parameters in LPUART_InitStruct.
Parameters	• **LPUARTx:** LPUART Instance • **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure that contains the configuration information for the specified LPUART peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: LPUART registers are initialized according to LPUART_InitStruct content – ERROR: Problem occurred during LPUART Registers initialization

Notes	• As some bits in LPUART configuration registers can only be written when the LPUART is disabled (USART_CR1_UE bit =0), LPUART IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned. • Baud rate value stored in LPUART_InitStruct BaudRate field, should be valid (different from 0).

LL_LPUART_StructInit

Function name	**void LL_LPUART_StructInit (LL_LPUART_InitTypeDef * LPUART_InitStruct)**
Function description	Set each LL_LPUART_InitTypeDef field to default value.
Parameters	• **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

88.3 LPUART Firmware driver defines

88.3.1 LPUART

Address Length Detection

LL_LPUART_ADDRESS_DETECT_4B	4-bit address detection method selected
LL_LPUART_ADDRESS_DETECT_7B	7-bit address detection (in 8-bit data mode) method selected

Binary Data Inversion

LL_LPUART_BINARY_LOGIC_POSITIVE	Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)
LL_LPUART_BINARY_LOGIC_NEGATIVE	Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_LPUART_BITORDER_LSBFIRST	data is transmitted/received with data bit 0 first, following the start bit
LL_LPUART_BITORDER_MSBFIRST	data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_LPUART_ICR_PECF	Parity error flag
LL_LPUART_ICR_FECF	Framing error flag
LL_LPUART_ICR_NCF	Noise detected flag
LL_LPUART_ICR_ORECF	Overrun error flag
LL_LPUART_ICR_IDLECF	Idle line detected flag
LL_LPUART_ICR_TXFECF	TX FIFO Empty Clear flag
LL_LPUART_ICR_TCCF	Transmission complete flag

LL_LPUART_ICR_CTSCF	CTS flag
LL_LPUART_ICR_CMCF	Character match flag
LL_LPUART_ICR_WUCF	Wakeup from Stop mode flag

Datawidth

LL_LPUART_DATAWIDTH_7B	7 bits word length: Start bit, 7 data bits, n stop bits
LL_LPUART_DATAWIDTH_8B	8 bits word length: Start bit, 8 data bits, n stop bits
LL_LPUART_DATAWIDTH_9B	9 bits word length: Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_LPUART_DE_POLARITY_HIGH	DE signal is active high
LL_LPUART_DE_POLARITY_LOW	DE signal is active low

Direction

LL_LPUART_DIRECTION_NONE	Transmitter and Receiver are disabled
LL_LPUART_DIRECTION_RX	Transmitter is disabled and Receiver is enabled
LL_LPUART_DIRECTION_TX	Transmitter is enabled and Receiver is disabled
LL_LPUART_DIRECTION_TX_RX	Transmitter and Receiver are enabled

DMA Register Data

LL_LPUART_DMA_REG_DATA_TRANSMIT	Get address of data register used for transmission
LL_LPUART_DMA_REG_DATA_RECEIVE	Get address of data register used for reception

FIFO Threshold

LL_LPUART_FIFOTHRESHOLD_1_8	FIFO reaches 1/8 of its depth
LL_LPUART_FIFOTHRESHOLD_1_4	FIFO reaches 1/4 of its depth
LL_LPUART_FIFOTHRESHOLD_1_2	FIFO reaches 1/2 of its depth
LL_LPUART_FIFOTHRESHOLD_3_4	FIFO reaches 3/4 of its depth
LL_LPUART_FIFOTHRESHOLD_7_8	FIFO reaches 7/8 of its depth
LL_LPUART_FIFOTHRESHOLD_8_8	FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_LPUART_ISR_PE	Parity error flag
LL_LPUART_ISR_FE	Framing error flag
LL_LPUART_ISR_NE	Noise detected flag
LL_LPUART_ISR_ORE	Overrun error flag
LL_LPUART_ISR_IDLE	Idle line detected flag
LL_LPUART_ISR_RXNE_RXFNE	Read data register or RX FIFO not empty flag
LL_LPUART_ISR_TC	Transmission complete flag
LL_LPUART_ISR_TXE_TXFNF	Transmit data register empty or TX FIFO Not Full flag
LL_LPUART_ISR_CTSIF	CTS interrupt flag

LL_LPUART_ISR_CTS	CTS flag
LL_LPUART_ISR_BUSY	Busy flag
LL_LPUART_ISR_CMF	Character match flag
LL_LPUART_ISR_SBKF	Send break flag
LL_LPUART_ISR_RWU	Receiver wakeup from Mute mode flag
LL_LPUART_ISR_WUF	Wakeup from Stop mode flag
LL_LPUART_ISR_TEACK	Transmit enable acknowledge flag
LL_LPUART_ISR_REACK	Receive enable acknowledge flag
LL_LPUART_ISR_TXFE	TX FIFO empty flag
LL_LPUART_ISR_RXFF	RX FIFO full flag
LL_LPUART_ISR_RXFT	RX FIFO threshold flag
LL_LPUART_ISR_TXFT	TX FIFO threshold flag

Hardware Control

LL_LPUART_HWCONTROL_NONE	CTS and RTS hardware flow control disabled
LL_LPUART_HWCONTROL_RTS	RTS output enabled, data is only requested when there is space in the receive buffer
LL_LPUART_HWCONTROL_CTS	CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)
LL_LPUART_HWCONTROL_RTS_CTS	CTS and RTS hardware flow control enabled

IT Defines

LL_LPUART_CR1_IDLEIE	IDLE interrupt enable
LL_LPUART_CR1_RXNEIE_RXFNEIE	Read data register and RXFIFO not empty interrupt enable
LL_LPUART_CR1_TCIE	Transmission complete interrupt enable
LL_LPUART_CR1_TXEIE_TXFNFIE	Transmit data register empty and TX FIFO not full interrupt enable
LL_LPUART_CR1_PEIE	Parity error
LL_LPUART_CR1_CMIE	Character match interrupt enable
LL_LPUART_CR1_TXFEIE	TX FIFO empty interrupt enable
LL_LPUART_CR1_RXFFIE	RX FIFO full interrupt enable
LL_LPUART_CR3_EIE	Error interrupt enable
LL_LPUART_CR3_CTSIE	CTS interrupt enable
LL_LPUART_CR3_WUFIE	Wakeup from Stop mode interrupt enable
LL_LPUART_CR3_TXFTIE	TX FIFO threshold interrupt enable
LL_LPUART_CR3_RXFTIE	RX FIFO threshold interrupt enable

Parity Control

LL_LPUART_PARITY_NONE	Parity control disabled
LL_LPUART_PARITY_EVEN	Parity control enabled and Even Parity is selected

LL_LPUART_PARITY_ODD	Parity control enabled and Odd Parity is selected

Clock Source Prescaler

LL_LPUART_PRESCALER_DIV1	Input clock not devided
LL_LPUART_PRESCALER_DIV2	Input clock devided by 2
LL_LPUART_PRESCALER_DIV4	Input clock devided by 4
LL_LPUART_PRESCALER_DIV6	Input clock devided by 6
LL_LPUART_PRESCALER_DIV8	Input clock devided by 8
LL_LPUART_PRESCALER_DIV10	Input clock devided by 10
LL_LPUART_PRESCALER_DIV12	Input clock devided by 12
LL_LPUART_PRESCALER_DIV16	Input clock devided by 16
LL_LPUART_PRESCALER_DIV32	Input clock devided by 32
LL_LPUART_PRESCALER_DIV64	Input clock devided by 64
LL_LPUART_PRESCALER_DIV128	Input clock devided by 128
LL_LPUART_PRESCALER_DIV256	Input clock devided by 256

RX Pin Active Level Inversion

LL_LPUART_RXPIN_LEVEL_STANDARD	RX pin signal works using the standard logic levels
LL_LPUART_RXPIN_LEVEL_INVERTED	RX pin signal values are inverted.

Stop Bits

LL_LPUART_STOPBITS_1	1 stop bit
LL_LPUART_STOPBITS_2	2 stop bits

TX Pin Active Level Inversion

LL_LPUART_TXPIN_LEVEL_STANDARD	TX pin signal works using the standard logic levels
LL_LPUART_TXPIN_LEVEL_INVERTED	TX pin signal values are inverted.

TX RX Pins Swap

LL_LPUART_TXRX_STANDARD	TX/RX pins are used as defined in standard pinout
LL_LPUART_TXRX_SWAPPED	TX and RX pins functions are swapped.

Wakeup

LL_LPUART_WAKEUP_IDLELINE	LPUART wake up from Mute mode on Idle Line
LL_LPUART_WAKEUP_ADDRESSMARK	LPUART wake up from Mute mode on Address Mark

Wakeup Activation

LL_LPUART_WAKEUP_ON_ADDRESS	Wake up active on address match
LL_LPUART_WAKEUP_ON_STARTBIT	Wake up active on Start bit detection
LL_LPUART_WAKEUP_ON_RXNE	Wake up active on RXNE

FLAG_Management

LL_LPUART_IsActiveFlag_RXNE

LL_LPUART_IsActiveFlag_TXE

IT_Management

LL_LPUART_EnableIT_RXNE

LL_LPUART_EnableIT_TXE

LL_LPUART_DisableIT_RXNE

LL_LPUART_DisableIT_TXE

LL_LPUART_IsEnabledIT_RXNE

LL_LPUART_IsEnabledIT_TXE

Helper Macros

__LL_LPUART_DIV

Description:

- Compute LPUARTDIV value according to Peripheral Clock and expected Baud Rate (20-bit value of LPUARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for LPUART Instance
- __BAUDRATE__: Baud Rate value to achieve

Return value:

- LPUARTDIV: value to be used for BRR register filling

Common Write and read registers Macros

LL_LPUART_WriteReg

Description:

- Write a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPUART_ReadReg

Description:

- Read a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be read

Return value:

- Register: value

89 LL OPAMP Generic Driver

89.1 OPAMP Firmware driver registers structures

89.1.1 LL_OPAMP_InitTypeDef

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t FunctionalMode***
- ***uint32_t InputNonInverting***
- ***uint32_t InputInverting***

Field Documentation

- ***uint32_t LL_OPAMP_InitTypeDef::PowerMode***
  Set OPAMP power mode. This parameter can be a value of ***OPAMP_LL_EC_POWERMODE***This feature can be modified afterwards using unitary function **LL_OPAMP_SetPowerMode()**.
- ***uint32_t LL_OPAMP_InitTypeDef::FunctionalMode***
  Set OPAMP functional mode by setting internal connections: OPAMP operation in standalone, follower, ... This parameter can be a value of ***OPAMP_LL_EC_FUNCTIONAL_MODE***
  Note: If OPAMP is configured in mode PGA, the gain can be configured using function **LL_OPAMP_SetPGAGain()**. This feature can be modified afterwards using unitary function **LL_OPAMP_SetFunctionalMode()**.
- ***uint32_t LL_OPAMP_InitTypeDef::InputNonInverting***
  Set OPAMP input non-inverting connection. This parameter can be a value of ***OPAMP_LL_EC_INPUT_NONINVERTING***This feature can be modified afterwards using unitary function **LL_OPAMP_SetInputNonInverting()**.
- ***uint32_t LL_OPAMP_InitTypeDef::InputInverting***
  Set OPAMP inverting input connection. This parameter can be a value of ***OPAMP_LL_EC_INPUT_INVERTING***
  Note:OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin), this parameter is discarded. This feature can be modified afterwards using unitary function **LL_OPAMP_SetInputInverting()**.

89.2 OPAMP Firmware driver API description

89.2.1 Detailed description of functions

LL_OPAMP_SetCommonPowerRange

Function name	**__STATIC_INLINE void LL_OPAMP_SetCommonPowerRange (OPAMP_Common_TypeDef * OPAMPxy_COMMON, uint32_t PowerRange)**
Function description	Set OPAMP power range.
Parameters	• **OPAMPxy_COMMON:** OPAMP common instance (can be set directly from CMSIS definition or by using helper macro __LL_OPAMP_COMMON_INSTANCE())

	• **PowerRange:** This parameter can be one of the following values: – LL_OPAMP_POWERSUPPLY_RANGE_LOW – LL_OPAMP_POWERSUPPLY_RANGE_HIGH
Return values	• **None:**
Notes	• The OPAMP power range applies to several OPAMP instances (if several OPAMP instances available on the selected device). • On this STM32 serie, setting of this feature is conditioned to OPAMP state: All OPAMP instances of the OPAMP common group must be disabled. This check can be done with function LL_OPAMP_IsEnabled() for each OPAMP instance or by using helper macro __LL_OPAMP_IS_ENABLED_ALL_COMMON_INSTANCE().
Reference Manual to LL API cross reference:	• CSR OPARANGE LL_OPAMP_SetCommonPowerRange

LL_OPAMP_GetCommonPowerRange

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_GetCommonPowerRange (OPAMP_Common_TypeDef * OPAMPxy_COMMON)**
Function description	Get OPAMP power range.
Parameters	• **OPAMPxy_COMMON:** OPAMP common instance (can be set directly from CMSIS definition or by using helper macro __LL_OPAMP_COMMON_INSTANCE())
Return values	• **Returned:** value can be one of the following values: – LL_OPAMP_POWERSUPPLY_RANGE_LOW – LL_OPAMP_POWERSUPPLY_RANGE_HIGH
Notes	• The OPAMP power range applies to several OPAMP instances (if several OPAMP instances available on the selected device).
Reference Manual to LL API cross reference:	• CSR OPARANGE LL_OPAMP_GetCommonPowerRange

LL_OPAMP_SetPowerMode

Function name	**__STATIC_INLINE void LL_OPAMP_SetPowerMode (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode)**
Function description	Set OPAMP power mode.
Parameters	• **OPAMPx:** OPAMP instance • **PowerMode:** This parameter can be one of the following values: – LL_OPAMP_POWERMODE_NORMAL – LL_OPAMP_POWERMODE_LOWPOWER
Return values	• **None:**

Notes	• The OPAMP must be disabled to change this configuration.
Reference Manual to LL API cross reference:	• CSR OPALPM LL_OPAMP_SetPowerMode

LL_OPAMP_GetPowerMode

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_GetPowerMode (OPAMP_TypeDef * OPAMPx)**
Function description	Get OPAMP power mode.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **Returned:** value can be one of the following values: – LL_OPAMP_POWERMODE_NORMAL – LL_OPAMP_POWERMODE_LOWPOWER
Reference Manual to LL API cross reference:	• CSR OPALPM LL_OPAMP_GetPowerMode

LL_OPAMP_SetMode

Function name	**__STATIC_INLINE void LL_OPAMP_SetMode (OPAMP_TypeDef * OPAMPx, uint32_t Mode)**
Function description	Set OPAMP mode calibration or functional.
Parameters	• **OPAMPx:** OPAMP instance • **Mode:** This parameter can be one of the following values: – LL_OPAMP_MODE_FUNCTIONAL – LL_OPAMP_MODE_CALIBRATION
Return values	• **None:**
Notes	• OPAMP mode corresponds to functional or calibration mode: functional mode: OPAMP operation in standalone, follower, ... Set functional mode using function LL_OPAMP_SetFunctionalMode().calibration mode: offset calibration of the selected transistors differential pair NMOS or PMOS. • On this STM32 serie, during calibration, OPAMP functional mode must be set to standalone or follower mode (in order to open internal connections to resistors of PGA mode). Refer to function LL_OPAMP_SetFunctionalMode().
Reference Manual to LL API cross reference:	• CSR CALON LL_OPAMP_SetMode

LL_OPAMP_GetMode

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_GetMode (OPAMP_TypeDef * OPAMPx)**
Function description	Get OPAMP mode calibration or functional.

Parameters	• **OPAMPx:** OPAMP instance
Return values	• **Returned:** value can be one of the following values: – LL_OPAMP_MODE_FUNCTIONAL – LL_OPAMP_MODE_CALIBRATION
Notes	• OPAMP mode corresponds to functional or calibration mode: functional mode: OPAMP operation in standalone, follower, ... Set functional mode using function LL_OPAMP_SetFunctionalMode().calibration mode: offset calibration of the selected transistors differential pair NMOS or PMOS.
Reference Manual to LL API cross reference:	• CSR CALON LL_OPAMP_GetMode

LL_OPAMP_SetFunctionalMode

Function name	**__STATIC_INLINE void LL_OPAMP_SetFunctionalMode (OPAMP_TypeDef * OPAMPx, uint32_t FunctionalMode)**
Function description	Set OPAMP functional mode by setting internal connections.
Parameters	• **OPAMPx:** OPAMP instance • **FunctionalMode:** This parameter can be one of the following values: – LL_OPAMP_MODE_STANDALONE – LL_OPAMP_MODE_FOLLOWER – LL_OPAMP_MODE_PGA
Return values	• **None:**
Notes	• This function reset bit of calibration mode to ensure to be in functional mode, in order to have OPAMP parameters (inputs selection, ...) set with the corresponding OPAMP mode to be effective.
Reference Manual to LL API cross reference:	• CSR OPAMODE LL_OPAMP_SetFunctionalMode

LL_OPAMP_GetFunctionalMode

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_GetFunctionalMode (OPAMP_TypeDef * OPAMPx)**
Function description	Get OPAMP functional mode from setting of internal connections.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **Returned:** value can be one of the following values: – LL_OPAMP_MODE_STANDALONE – LL_OPAMP_MODE_FOLLOWER – LL_OPAMP_MODE_PGA
Reference Manual to LL API cross reference:	• CSR OPAMODE LL_OPAMP_GetFunctionalMode

LL_OPAMP_SetPGAGain

Function name	**__STATIC_INLINE void LL_OPAMP_SetPGAGain (OPAMP_TypeDef * OPAMPx, uint32_t PGAGain)**
Function description	Set OPAMP PGA gain.
Parameters	• **OPAMPx:** OPAMP instance • **PGAGain:** This parameter can be one of the following values: – LL_OPAMP_PGA_GAIN_2 – LL_OPAMP_PGA_GAIN_4 – LL_OPAMP_PGA_GAIN_8 – LL_OPAMP_PGA_GAIN_16
Return values	• **None:**
Notes	• Preliminarily, OPAMP must be set in mode PGA using function LL_OPAMP_SetFunctionalMode().
Reference Manual to LL API cross reference:	• CSR PGGAIN LL_OPAMP_SetPGAGain

LL_OPAMP_GetPGAGain

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_GetPGAGain (OPAMP_TypeDef * OPAMPx)**
Function description	Get OPAMP PGA gain.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **Returned:** value can be one of the following values: – LL_OPAMP_PGA_GAIN_2 – LL_OPAMP_PGA_GAIN_4 – LL_OPAMP_PGA_GAIN_8 – LL_OPAMP_PGA_GAIN_16
Notes	• Preliminarily, OPAMP must be set in mode PGA using function LL_OPAMP_SetFunctionalMode().
Reference Manual to LL API cross reference:	• CSR PGGAIN LL_OPAMP_GetPGAGain

LL_OPAMP_SetInputNonInverting

Function name	**__STATIC_INLINE void LL_OPAMP_SetInputNonInverting (OPAMP_TypeDef * OPAMPx, uint32_t InputNonInverting)**
Function description	Set OPAMP non-inverting input connection.
Parameters	• **OPAMPx:** OPAMP instance • **InputNonInverting:** This parameter can be one of the following values: – LL_OPAMP_INPUT_NONINVERT_IO0 – LL_OPAMP_INPUT_NONINV_DAC1_CH1
Return values	• **None:**

Reference Manual to LL API cross reference:	• CSR VPSEL LL_OPAMP_SetInputNonInverting

LL_OPAMP_GetInputNonInverting

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_GetInputNonInverting (OPAMP_TypeDef * OPAMPx)**
Function description	Get OPAMP non-inverting input connection.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **Returned:** value can be one of the following values: – LL_OPAMP_INPUT_NONINVERT_IO0 – LL_OPAMP_INPUT_NONINV_DAC1_CH1
Reference Manual to LL API cross reference:	• CSR VPSEL LL_OPAMP_GetInputNonInverting

LL_OPAMP_SetInputInverting

Function name	**__STATIC_INLINE void LL_OPAMP_SetInputInverting (OPAMP_TypeDef * OPAMPx, uint32_t InputInverting)**
Function description	Set OPAMP inverting input connection.
Parameters	• **OPAMPx:** OPAMP instance • **InputInverting:** This parameter can be one of the following values: – LL_OPAMP_INPUT_INVERT_IO0 – LL_OPAMP_INPUT_INVERT_IO1 – LL_OPAMP_INPUT_INVERT_CONNECT_NO
Return values	• **None:**
Notes	• OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin).
Reference Manual to LL API cross reference:	• CSR VMSEL LL_OPAMP_SetInputInverting

LL_OPAMP_GetInputInverting

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_GetInputInverting (OPAMP_TypeDef * OPAMPx)**
Function description	Get OPAMP inverting input connection.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **Returned:** value can be one of the following values: – LL_OPAMP_INPUT_INVERT_IO0 – LL_OPAMP_INPUT_INVERT_IO1 – LL_OPAMP_INPUT_INVERT_CONNECT_NO

Reference Manual to LL API cross reference:	• CSR VMSEL LL_OPAMP_GetInputInverting

LL_OPAMP_SetNonInvertingInput

Function name	**__STATIC_INLINE void LL_OPAMP_SetNonInvertingInput (OPAMP_TypeDef * OPAMPx, uint32_t NonInvertingInput)**
Function description	

LL_OPAMP_SetInvertingInput

Function name	**__STATIC_INLINE void LL_OPAMP_SetInvertingInput (OPAMP_TypeDef * OPAMPx, uint32_t InvertingInput)**
Function description	

LL_OPAMP_SetTrimmingMode

Function name	**__STATIC_INLINE void LL_OPAMP_SetTrimmingMode (OPAMP_TypeDef * OPAMPx, uint32_t TrimmingMode)**
Function description	Set OPAMP trimming mode.
Parameters	• **OPAMPx:** OPAMP instance • **TrimmingMode:** This parameter can be one of the following values: – LL_OPAMP_TRIMMING_FACTORY – LL_OPAMP_TRIMMING_USER
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR USERTRIM LL_OPAMP_SetTrimmingMode

LL_OPAMP_GetTrimmingMode

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_GetTrimmingMode (OPAMP_TypeDef * OPAMPx)**
Function description	Get OPAMP trimming mode.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **Returned:** value can be one of the following values: – LL_OPAMP_TRIMMING_FACTORY – LL_OPAMP_TRIMMING_USER
Reference Manual to LL API cross reference:	• CSR USERTRIM LL_OPAMP_GetTrimmingMode

LL_OPAMP_SetCalibrationSelection

Function name	**__STATIC_INLINE void LL_OPAMP_SetCalibrationSelection (OPAMP_TypeDef * OPAMPx, uint32_t TransistorsDiffPair)**

Function description	Set OPAMP offset to calibrate the selected transistors differential pair NMOS or PMOS.
Parameters	• **OPAMPx:** OPAMP instance • **TransistorsDiffPair:** This parameter can be one of the following values: – LL_OPAMP_TRIMMING_NMOS – LL_OPAMP_TRIMMING_PMOS
Return values	• **None:**
Notes	• Preliminarily, OPAMP must be set in mode calibration using function LL_OPAMP_SetMode().
Reference Manual to LL API cross reference:	• CSR CALSEL LL_OPAMP_SetCalibrationSelection

LL_OPAMP_GetCalibrationSelection

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_GetCalibrationSelection (OPAMP_TypeDef * OPAMPx)**
Function description	Get OPAMP offset to calibrate the selected transistors differential pair NMOS or PMOS.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **Returned:** value can be one of the following values: – LL_OPAMP_TRIMMING_NMOS – LL_OPAMP_TRIMMING_PMOS
Notes	• Preliminarily, OPAMP must be set in mode calibration using function LL_OPAMP_SetMode().
Reference Manual to LL API cross reference:	• CSR CALSEL LL_OPAMP_GetCalibrationSelection

LL_OPAMP_IsCalibrationOutputSet

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_IsCalibrationOutputSet (OPAMP_TypeDef * OPAMPx)**
Function description	Get OPAMP calibration result of toggling output.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **State:** of bit (1 or 0).
Notes	• This functions returns: 0 if OPAMP calibration output is reset 1 if OPAMP calibration output is set
Reference Manual to LL API cross reference:	• CSR CALOUT LL_OPAMP_IsCalibrationOutputSet

LL_OPAMP_SetTrimmingValue

Function name	**__STATIC_INLINE void LL_OPAMP_SetTrimmingValue (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode, uint32_t TransistorsDiffPair, uint32_t TrimmingValue)**
Function description	Set OPAMP trimming factor for the selected transistors differential pair NMOS or PMOS, corresponding to the selected power mode.
Parameters	• **OPAMPx:** OPAMP instance • **PowerMode:** This parameter can be one of the following values: – LL_OPAMP_POWERMODE_NORMAL – LL_OPAMP_POWERMODE_LOWPOWER • **TransistorsDiffPair:** This parameter can be one of the following values: – LL_OPAMP_TRIMMING_NMOS – LL_OPAMP_TRIMMING_PMOS • **TrimmingValue:** 0x00...0x1F
Return values	• **None:**
Reference Manual to LL API cross reference:	• OTR TRIMOFFSETN LL_OPAMP_SetTrimmingValue • OTR TRIMOFFSETP LL_OPAMP_SetTrimmingValue • LPOTR TRIMLPOFFSETN LL_OPAMP_SetTrimmingValue • LPOTR TRIMLPOFFSETP LL_OPAMP_SetTrimmingValue

LL_OPAMP_GetTrimmingValue

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_GetTrimmingValue (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode, uint32_t TransistorsDiffPair)**
Function description	Get OPAMP trimming factor for the selected transistors differential pair NMOS or PMOS, corresponding to the selected power mode.
Parameters	• **OPAMPx:** OPAMP instance • **PowerMode:** This parameter can be one of the following values: – LL_OPAMP_POWERMODE_NORMAL – LL_OPAMP_POWERMODE_LOWPOWER • **TransistorsDiffPair:** This parameter can be one of the following values: – LL_OPAMP_TRIMMING_NMOS – LL_OPAMP_TRIMMING_PMOS
Return values	• **0x0...0x1F:**
Reference Manual to LL API cross reference:	• OTR TRIMOFFSETN LL_OPAMP_GetTrimmingValue • OTR TRIMOFFSETP LL_OPAMP_GetTrimmingValue • LPOTR TRIMLPOFFSETN LL_OPAMP_GetTrimmingValue • LPOTR TRIMLPOFFSETP LL_OPAMP_GetTrimmingValue

LL_OPAMP_Enable

Function name	**__STATIC_INLINE void LL_OPAMP_Enable (OPAMP_TypeDef * OPAMPx)**

Function description	Enable OPAMP instance.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **None:**
Notes	• After enable from off state, OPAMP requires a delay to fullfill wake up time specification. Refer to device datasheet, parameter "tWAKEUP".
Reference Manual to LL API cross reference:	• CSR OPAMPXEN LL_OPAMP_Enable

LL_OPAMP_Disable

Function name	**__STATIC_INLINE void LL_OPAMP_Disable (OPAMP_TypeDef * OPAMPx)**
Function description	Disable OPAMP instance.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR OPAMPXEN LL_OPAMP_Disable

LL_OPAMP_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_OPAMP_IsEnabled (OPAMP_TypeDef * OPAMPx)**
Function description	Get OPAMP instance enable state (0: OPAMP is disabled, 1: OPAMP is enabled)
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR OPAMPXEN LL_OPAMP_IsEnabled

LL_OPAMP_DeInit

Function name	**ErrorStatus LL_OPAMP_DeInit (OPAMP_TypeDef * OPAMPx)**
Function description	De-initialize registers of the selected OPAMP instance to their default reset values.
Parameters	• **OPAMPx:** OPAMP instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: OPAMP registers are de-initialized – ERROR: OPAMP registers are not de-initialized

LL_OPAMP_Init

Function name	**ErrorStatus LL_OPAMP_Init (OPAMP_TypeDef * OPAMPx,**

	LL_OPAMP_InitTypeDef * OPAMP_InitStruct)
Function description	Initialize some features of OPAMP instance.
Parameters	• **OPAMPx:** OPAMP instance • **OPAMP_InitStruct:** Pointer to a LL_OPAMP_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: OPAMP registers are initialized – ERROR: OPAMP registers are not initialized
Notes	• This function reset bit of calibration mode to ensure to be in functional mode, in order to have OPAMP parameters (inputs selection, ...) set with the corresponding OPAMP mode to be effective. • This function configures features of the selected OPAMP instance. Some features are also available at scope OPAMP common instance (common to several OPAMP instances). Refer to functions having argument "OPAMPxy_COMMON" as parameter.

LL_OPAMP_StructInit

Function name	**void LL_OPAMP_StructInit (LL_OPAMP_InitTypeDef * OPAMP_InitStruct)**
Function description	Set each LL_OPAMP_InitTypeDef field to default value.
Parameters	• **OPAMP_InitStruct:** pointer to a LL_OPAMP_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

89.3 OPAMP Firmware driver defines

89.3.1 OPAMP

OPAMP functional mode

LL_OPAMP_MODE_STANDALONE	OPAMP functional mode, OPAMP operation in standalone
LL_OPAMP_MODE_FOLLOWER	OPAMP functional mode, OPAMP operation in follower
LL_OPAMP_MODE_PGA	OPAMP functional mode, OPAMP operation in PGA

Definitions of OPAMP hardware constraints delays

LL_OPAMP_DELAY_STARTUP_US	Delay for OPAMP startup time

OPAMP input inverting

LL_OPAMP_INPUT_INVERT_IO0	OPAMP inverting input connected to GPIO pin (valid also in PGA mode for filtering). Note: OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not

	used (not connected to GPIO pin).
LL_OPAMP_INPUT_INVERT_IO1	OPAMP inverting input (low leakage input) connected to GPIO pin (available only on package BGA132). Note: OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin).
LL_OPAMP_INPUT_INVERT_CONNECT_NO	OPAMP inverting input not externally connected (intended for OPAMP in mode follower or PGA without external capacitors for filtering)

OPAMP inputs legacy literals name

LL_OPAMP_NONINVERTINGINPUT_IO0

LL_OPAMP_NONINVERTINGINPUT_DAC_CH

LL_OPAMP_INVERTINGINPUT_IO0

LL_OPAMP_INVERTINGINPUT_IO1

LL_OPAMP_INVERTINGINPUT_CONNECT_NO

LL_OPAMP_INPUT_NONINVERT_DAC1_CH1

OPAMP input non-inverting

LL_OPAMP_INPUT_NONINVERT_IO0	OPAMP non inverting input connected to GPIO pin (pin PA0 for OPAMP1, pin PA6 for OPAMP2)
LL_OPAMP_INPUT_NONINV_DAC1_CH1	OPAMP non inverting input connected to DAC1 channel1 output

OPAMP mode calibration or functional.

LL_OPAMP_MODE_FUNCTIONAL	OPAMP functional mode
LL_OPAMP_MODE_CALIBRATION	OPAMP calibration mode

OPAMP PGA gain (relevant when OPAMP is in functional mode PGA)

LL_OPAMP_PGA_GAIN_2	OPAMP PGA gain 2
LL_OPAMP_PGA_GAIN_4	OPAMP PGA gain 4
LL_OPAMP_PGA_GAIN_8	OPAMP PGA gain 8
LL_OPAMP_PGA_GAIN_16	OPAMP PGA gain 16

OPAMP power mode

LL_OPAMP_POWERMODE_NORMAL	OPAMP power mode normal
LL_OPAMP_POWERMODE_LOWPOWER	OPAMP power mode low-power

OPAMP power supply range

LL_OPAMP_POWERSUPPLY_RANGE_LOW	Power supply range low. On STM32L4 serie: Vdda lower than 2.4V.

LL_OPAMP_POWERSUPPLY_RANGE_HIGH	Power supply range high. On STM32L4 serie: Vdda higher than 2.4V.

OPAMP trimming mode

LL_OPAMP_TRIMMING_FACTORY	OPAMP trimming factors set to factory values
LL_OPAMP_TRIMMING_USER	OPAMP trimming factors set to user values

OPAMP trimming of transistors differential pair NMOS or PMOS

LL_OPAMP_TRIMMING_NMOS	OPAMP trimming of transistors differential pair NMOS
LL_OPAMP_TRIMMING_PMOS	OPAMP trimming of transistors differential pair PMOS

OPAMP helper macro

__LL_OPAMP_COMMON_INSTANCE_

Description:

- Helper macro to select the OPAMP common instance to which is belonging the selected OPAMP instance.

Parameters:

- __OPAMPx__: OPAMP instance

Return value:

- OPAMP: common instance

Notes:

- OPAMP common register instance can be used to set parameters common to several OPAMP instances. Refer to functions having argument "OPAMPxy_COMMON" as parameter.

__LL_OPAMP_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all OPAMP instances sharing the same OPAMP common instance are disabled.

Return value:

- 0: All OPAMP instances sharing the same OPAMP common instance are disabled. 1: At least one OPAMP instance sharing the same OPAMP common instance is enabled

Notes:

- This check is required by functions with setting conditioned to OPAMP state:

All OPAMP instances of the OPAMP common group must be disabled. Refer to functions having argument "OPAMPxy_COMMON" as parameter.

Common write and read registers macro

LL_OPAMP_WriteReg

Description:

- Write a value in OPAMP register.

Parameters:

- __INSTANCE__: OPAMP Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_OPAMP_ReadReg

Description:

- Read a value in OPAMP register.

Parameters:

- __INSTANCE__: OPAMP Instance
- __REG__: Register to be read

Return value:

- Register: value

90 LL PWR Generic Driver

90.1 PWR Firmware driver API description

90.1.1 Detailed description of functions

LL_PWR_EnableLowPowerRunMode

Function name	**__STATIC_INLINE void LL_PWR_EnableLowPowerRunMode (void)**
Function description	Switch the regulator from main mode to low-power mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 LPR LL_PWR_EnableLowPowerRunMode

LL_PWR_DisableLowPowerRunMode

Function name	**__STATIC_INLINE void LL_PWR_DisableLowPowerRunMode (void)**
Function description	Switch the regulator from low-power mode to main mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 LPR LL_PWR_DisableLowPowerRunMode

LL_PWR_EnterLowPowerRunMode

Function name	**__STATIC_INLINE void LL_PWR_EnterLowPowerRunMode (void)**
Function description	Switch from run main mode to run low-power mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 LPR LL_PWR_EnterLowPowerRunMode

LL_PWR_ExitLowPowerRunMode

Function name	**__STATIC_INLINE void LL_PWR_ExitLowPowerRunMode (void)**
Function description	Switch from run main mode to low-power mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 LPR LL_PWR_ExitLowPowerRunMode

LL_PWR_IsEnabledLowPowerRunMode

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledLowPowerRunMode (void)**
Function description	Check if the regulator is in low-power mode.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 LPR LL_PWR_IsEnabledLowPowerRunMode

LL_PWR_SetRegulVoltageScaling

Function name	**__STATIC_INLINE void LL_PWR_SetRegulVoltageScaling (uint32_t VoltageScaling)**
Function description	Set the main internal regulator output voltage.
Parameters	• **VoltageScaling:** This parameter can be one of the following values: – LL_PWR_REGU_VOLTAGE_SCALE1 – LL_PWR_REGU_VOLTAGE_SCALE2
Return values	• **None:**
Notes	• This configuration may be completed with LL_PWR_EnableRange1BoostMode() on STM32L4Rx/STM32L4Sx devices.
Reference Manual to LL API cross reference:	• CR1 VOS LL_PWR_SetRegulVoltageScaling

LL_PWR_GetRegulVoltageScaling

Function name	**__STATIC_INLINE uint32_t LL_PWR_GetRegulVoltageScaling (void)**
Function description	Get the main internal regulator output voltage.
Return values	• **Returned:** value can be one of the following values: – LL_PWR_REGU_VOLTAGE_SCALE1 – LL_PWR_REGU_VOLTAGE_SCALE2
Reference Manual to LL API cross reference:	• CR1 VOS LL_PWR_GetRegulVoltageScaling

LL_PWR_EnableRange1BoostMode

Function name	**__STATIC_INLINE void LL_PWR_EnableRange1BoostMode (void)**
Function description	Enable main regulator voltage range 1 boost mode.
Return values	• **None:**
Reference Manual to LL API cross	• CR5 R1MODE LL_PWR_EnableRange1BoostMode

reference:

LL_PWR_DisableRange1BoostMode

Function name	**__STATIC_INLINE void LL_PWR_DisableRange1BoostMode (void)**
Function description	Disable main regulator voltage range 1 boost mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR5 R1MODE LL_PWR_DisableRange1BoostMode

LL_PWR_IsEnabledRange1BoostMode

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledRange1BoostMode (void)**
Function description	Check if the main regulator voltage range 1 boost mode is enabled.
Return values	• **Inverted:** state of bit (0 or 1).
Reference Manual to LL API cross reference:	• CR5 R1MODE LL_PWR_IsEnabledRange1BoostMode

LL_PWR_EnableBkUpAccess

Function name	**__STATIC_INLINE void LL_PWR_EnableBkUpAccess (void)**
Function description	Enable access to the backup domain.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 DBP LL_PWR_EnableBkUpAccess

LL_PWR_DisableBkUpAccess

Function name	**__STATIC_INLINE void LL_PWR_DisableBkUpAccess (void)**
Function description	Disable access to the backup domain.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 DBP LL_PWR_DisableBkUpAccess

LL_PWR_IsEnabledBkUpAccess

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpAccess (void)**
Function description	Check if the backup domain is enabled.

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 DBP LL_PWR_IsEnabledBkUpAccess

LL_PWR_SetPowerMode

Function name	**__STATIC_INLINE void LL_PWR_SetPowerMode (uint32_t LowPowerMode)**
Function description	Set Low-Power mode.
Parameters	• **LowPowerMode:** This parameter can be one of the following values: – LL_PWR_MODE_STOP0 – LL_PWR_MODE_STOP1 – LL_PWR_MODE_STOP2 – LL_PWR_MODE_STANDBY – LL_PWR_MODE_SHUTDOWN
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 LPMS LL_PWR_SetPowerMode

LL_PWR_GetPowerMode

Function name	**__STATIC_INLINE uint32_t LL_PWR_GetPowerMode (void)**
Function description	Get Low-Power mode.
Return values	• **Returned:** value can be one of the following values: – LL_PWR_MODE_STOP0 – LL_PWR_MODE_STOP1 – LL_PWR_MODE_STOP2 – LL_PWR_MODE_STANDBY – LL_PWR_MODE_SHUTDOWN
Reference Manual to LL API cross reference:	• CR1 LPMS LL_PWR_GetPowerMode

LL_PWR_EnableSRAM3Retention

Function name	**__STATIC_INLINE void LL_PWR_EnableSRAM3Retention (void)**
Function description	Enable SRAM3 content retention in Stop mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RRSTP LL_PWR_EnableSRAM3Retention

LL_PWR_DisableSRAM3Retention

Function name	**__STATIC_INLINE void LL_PWR_DisableSRAM3Retention (void)**
Function description	Disable SRAM3 content retention in Stop mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RRSTP LL_PWR_DisableSRAM3Retention

LL_PWR_IsEnabledSRAM3Retention

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledSRAM3Retention (void)**
Function description	Check if SRAM3 content retention in Stop mode is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RRSTP LL_PWR_IsEnabledSRAM3Retention

LL_PWR_EnableDSIPinsPDActivation

Function name	**__STATIC_INLINE void LL_PWR_EnableDSIPinsPDActivation (void)**
Function description	Enable pull-down activation on DSI pins.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DSIPDEN LL_PWR_EnableDSIPinsPDActivation

LL_PWR_DisableDSIPinsPDActivation

Function name	**__STATIC_INLINE void LL_PWR_DisableDSIPinsPDActivation (void)**
Function description	Disable pull-down activation on DSI pins.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DSIPDEN LL_PWR_DisableDSIPinsPDActivation

LL_PWR_IsEnabledDSIPinsPDActivation

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledDSIPinsPDActivation (void)**
Function description	Check if pull-down activation on DSI pins is enabled.
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• CR3 DSIPDEN LL_PWR_IsEnabledDSIPinsPDActivation

LL_PWR_EnableVddUSB

Function name	**__STATIC_INLINE void LL_PWR_EnableVddUSB (void)**
Function description	Enable VDDUSB supply.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 USV LL_PWR_EnableVddUSB

LL_PWR_DisableVddUSB

Function name	**__STATIC_INLINE void LL_PWR_DisableVddUSB (void)**
Function description	Disable VDDUSB supply.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 USV LL_PWR_DisableVddUSB

LL_PWR_IsEnabledVddUSB

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledVddUSB (void)**
Function description	Check if VDDUSB supply is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 USV LL_PWR_IsEnabledVddUSB

LL_PWR_EnableVddIO2

Function name	**__STATIC_INLINE void LL_PWR_EnableVddIO2 (void)**
Function description	Enable VDDIO2 supply.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 IOSV LL_PWR_EnableVddIO2

LL_PWR_DisableVddIO2

Function name	**__STATIC_INLINE void LL_PWR_DisableVddIO2 (void)**
Function description	Disable VDDIO2 supply.
Return values	• **None:**
Reference Manual to	• CR2 IOSV LL_PWR_DisableVddIO2

LL API cross reference:

LL_PWR_IsEnabledVddIO2

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledVddIO2 (void)**
Function description	Check if VDDIO2 supply is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 IOSV LL_PWR_IsEnabledVddIO2

LL_PWR_EnablePVM

Function name	**__STATIC_INLINE void LL_PWR_EnablePVM (uint32_t PeriphVoltage)**
Function description	Enable the Power Voltage Monitoring on a peripheral.
Parameters	• **PeriphVoltage:** This parameter can be one of the following values: (*) value not defined in all devices – LL_PWR_PVM_VDDUSB_1_2V (*) – LL_PWR_PVM_VDDIO2_0_9V (*) – LL_PWR_PVM_VDDA_1_62V – LL_PWR_PVM_VDDA_2_2V
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 PVME1 LL_PWR_EnablePVM • CR2 PVME2 LL_PWR_EnablePVM • CR2 PVME3 LL_PWR_EnablePVM • CR2 PVME4 LL_PWR_EnablePVM

LL_PWR_DisablePVM

Function name	**__STATIC_INLINE void LL_PWR_DisablePVM (uint32_t PeriphVoltage)**
Function description	Disable the Power Voltage Monitoring on a peripheral.
Parameters	• **PeriphVoltage:** This parameter can be one of the following values: (*) value not defined in all devices – LL_PWR_PVM_VDDUSB_1_2V (*) – LL_PWR_PVM_VDDIO2_0_9V (*) – LL_PWR_PVM_VDDA_1_62V – LL_PWR_PVM_VDDA_2_2V
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 PVME1 LL_PWR_DisablePVM • CR2 PVME2 LL_PWR_DisablePVM • CR2 PVME3 LL_PWR_DisablePVM • CR2 PVME4 LL_PWR_DisablePVM

LL_PWR_IsEnabledPVM

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledPVM (uint32_t PeriphVoltage)**
Function description	Check if Power Voltage Monitoring is enabled on a peripheral.
Parameters	• **PeriphVoltage:** This parameter can be one of the following values: (*) value not defined in all devices – LL_PWR_PVM_VDDUSB_1_2V (*) – LL_PWR_PVM_VDDIO2_0_9V (*) – LL_PWR_PVM_VDDA_1_62V – LL_PWR_PVM_VDDA_2_2V
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 PVME1 LL_PWR_IsEnabledPVM • CR2 PVME2 LL_PWR_IsEnabledPVM • CR2 PVME3 LL_PWR_IsEnabledPVM • CR2 PVME4 LL_PWR_IsEnabledPVM

LL_PWR_SetPVDLevel

Function name	**__STATIC_INLINE void LL_PWR_SetPVDLevel (uint32_t PVDLevel)**
Function description	Configure the voltage threshold detected by the Power Voltage Detector.
Parameters	• **PVDLevel:** This parameter can be one of the following values: – LL_PWR_PVDLEVEL_0 – LL_PWR_PVDLEVEL_1 – LL_PWR_PVDLEVEL_2 – LL_PWR_PVDLEVEL_3 – LL_PWR_PVDLEVEL_4 – LL_PWR_PVDLEVEL_5 – LL_PWR_PVDLEVEL_6 – LL_PWR_PVDLEVEL_7
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 PLS LL_PWR_SetPVDLevel

LL_PWR_GetPVDLevel

Function name	**__STATIC_INLINE uint32_t LL_PWR_GetPVDLevel (void)**
Function description	Get the voltage threshold detection.
Return values	• **Returned:** value can be one of the following values: – LL_PWR_PVDLEVEL_0 – LL_PWR_PVDLEVEL_1 – LL_PWR_PVDLEVEL_2 – LL_PWR_PVDLEVEL_3 – LL_PWR_PVDLEVEL_4 – LL_PWR_PVDLEVEL_5

- – LL_PWR_PVDLEVEL_6
- – LL_PWR_PVDLEVEL_7

Reference Manual to LL API cross reference:	• CR2 PLS LL_PWR_GetPVDLevel

LL_PWR_EnablePVD

Function name	**__STATIC_INLINE void LL_PWR_EnablePVD (void)**
Function description	Enable Power Voltage Detector.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 PVDE LL_PWR_EnablePVD

LL_PWR_DisablePVD

Function name	**__STATIC_INLINE void LL_PWR_DisablePVD (void)**
Function description	Disable Power Voltage Detector.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 PVDE LL_PWR_DisablePVD

LL_PWR_IsEnabledPVD

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledPVD (void)**
Function description	Check if Power Voltage Detector is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 PVDE LL_PWR_IsEnabledPVD

LL_PWR_EnableInternWU

Function name	**__STATIC_INLINE void LL_PWR_EnableInternWU (void)**
Function description	Enable Internal Wake-up line.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 EIWF LL_PWR_EnableInternWU

LL_PWR_DisableInternWU

Function name	**__STATIC_INLINE void LL_PWR_DisableInternWU (void)**
Function description	Disable Internal Wake-up line.

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 EIWF LL_PWR_DisableInternWU

LL_PWR_IsEnabledInternWU

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledInternWU (void)**
Function description	Check if Internal Wake-up line is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 EIWF LL_PWR_IsEnabledInternWU

LL_PWR_EnablePUPDCfg

Function name	**__STATIC_INLINE void LL_PWR_EnablePUPDCfg (void)**
Function description	Enable pull-up and pull-down configuration.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 APC LL_PWR_EnablePUPDCfg

LL_PWR_DisablePUPDCfg

Function name	**__STATIC_INLINE void LL_PWR_DisablePUPDCfg (void)**
Function description	Disable pull-up and pull-down configuration.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 APC LL_PWR_DisablePUPDCfg

LL_PWR_IsEnabledPUPDCfg

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledPUPDCfg (void)**
Function description	Check if pull-up and pull-down configuration is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 APC LL_PWR_IsEnabledPUPDCfg

LL_PWR_EnableDSIPullDown

Function name	**__STATIC_INLINE void LL_PWR_EnableDSIPullDown (void)**

Function description	Enable pull-down activation on DSI pins.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DSIPDEN LL_PWR_EnableDSIPullDown

LL_PWR_DisableDSIPullDown

Function name	**__STATIC_INLINE void LL_PWR_DisableDSIPullDown (void)**
Function description	Disable pull-down activation on DSI pins.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DSIPDEN LL_PWR_DisableDSIPullDown

LL_PWR_IsEnabledDSIPullDown

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledDSIPullDown (void)**
Function description	Check if pull-down activation on DSI pins is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DSIPDEN LL_PWR_IsEnabledDSIPullDown

LL_PWR_EnableSRAM2Retention

Function name	**__STATIC_INLINE void LL_PWR_EnableSRAM2Retention (void)**
Function description	Enable SRAM2 content retention in Standby mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 RRS LL_PWR_EnableSRAM2Retention

LL_PWR_DisableSRAM2Retention

Function name	**__STATIC_INLINE void LL_PWR_DisableSRAM2Retention (void)**
Function description	Disable SRAM2 content retention in Standby mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 RRS LL_PWR_DisableSRAM2Retention

LL_PWR_IsEnabledSRAM2Retention

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledSRAM2Retention (void)**
Function description	Check if SRAM2 content retention in Standby mode is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 RRS LL_PWR_IsEnabledSRAM2Retention

LL_PWR_EnableWakeUpPin

Function name	**__STATIC_INLINE void LL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)**
Function description	Enable the WakeUp PINx functionality.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: – LL_PWR_WAKEUP_PIN1 – LL_PWR_WAKEUP_PIN2 – LL_PWR_WAKEUP_PIN3 – LL_PWR_WAKEUP_PIN4 – LL_PWR_WAKEUP_PIN5
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 EWUP1 LL_PWR_EnableWakeUpPin • CR3 EWUP2 LL_PWR_EnableWakeUpPin • CR3 EWUP3 LL_PWR_EnableWakeUpPin • CR3 EWUP4 LL_PWR_EnableWakeUpPin • CR3 EWUP5 LL_PWR_EnableWakeUpPin •

LL_PWR_DisableWakeUpPin

Function name	**__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)**
Function description	Disable the WakeUp PINx functionality.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: – LL_PWR_WAKEUP_PIN1 – LL_PWR_WAKEUP_PIN2 – LL_PWR_WAKEUP_PIN3 – LL_PWR_WAKEUP_PIN4 – LL_PWR_WAKEUP_PIN5
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 EWUP1 LL_PWR_DisableWakeUpPin • CR3 EWUP2 LL_PWR_DisableWakeUpPin • CR3 EWUP3 LL_PWR_DisableWakeUpPin • CR3 EWUP4 LL_PWR_DisableWakeUpPin • CR3 EWUP5 LL_PWR_DisableWakeUpPin

•

LL_PWR_IsEnabledWakeUpPin

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledWakeUpPin (uint32_t WakeUpPin)**
Function description	Check if the WakeUp PINx functionality is enabled.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: – LL_PWR_WAKEUP_PIN1 – LL_PWR_WAKEUP_PIN2 – LL_PWR_WAKEUP_PIN3 – LL_PWR_WAKEUP_PIN4 – LL_PWR_WAKEUP_PIN5
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 EWUP1 LL_PWR_IsEnabledWakeUpPin • CR3 EWUP2 LL_PWR_IsEnabledWakeUpPin • CR3 EWUP3 LL_PWR_IsEnabledWakeUpPin • CR3 EWUP4 LL_PWR_IsEnabledWakeUpPin • CR3 EWUP5 LL_PWR_IsEnabledWakeUpPin •

LL_PWR_SetBattChargResistor

Function name	**__STATIC_INLINE void LL_PWR_SetBattChargResistor (uint32_t Resistor)**
Function description	Set the resistor impedance.
Parameters	• **Resistor:** This parameter can be one of the following values: – LL_PWR_BATT_CHARG_RESISTOR_5K – LL_PWR_BATT_CHARGRESISTOR_1_5K
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR4 VBRS LL_PWR_SetBattChargResistor

LL_PWR_GetBattChargResistor

Function name	**__STATIC_INLINE uint32_t LL_PWR_GetBattChargResistor (void)**
Function description	Get the resistor impedance.
Return values	• **Returned:** value can be one of the following values: – LL_PWR_BATT_CHARG_RESISTOR_5K – LL_PWR_BATT_CHARGRESISTOR_1_5K
Reference Manual to LL API cross reference:	• CR4 VBRS LL_PWR_GetBattChargResistor

LL_PWR_EnableBatteryCharging

Function name	**__STATIC_INLINE void LL_PWR_EnableBatteryCharging (void)**
Function description	Enable battery charging.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR4 VBE LL_PWR_EnableBatteryCharging

LL_PWR_DisableBatteryCharging

Function name	**__STATIC_INLINE void LL_PWR_DisableBatteryCharging (void)**
Function description	Disable battery charging.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR4 VBE LL_PWR_DisableBatteryCharging

LL_PWR_IsEnabledBatteryCharging

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledBatteryCharging (void)**
Function description	Check if battery charging is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR4 VBE LL_PWR_IsEnabledBatteryCharging

LL_PWR_SetWakeUpPinPolarityLow

Function name	**__STATIC_INLINE void LL_PWR_SetWakeUpPinPolarityLow (uint32_t WakeUpPin)**
Function description	Set the Wake-Up pin polarity low for the event detection.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: – LL_PWR_WAKEUP_PIN1 – LL_PWR_WAKEUP_PIN2 – LL_PWR_WAKEUP_PIN3 – LL_PWR_WAKEUP_PIN4 – LL_PWR_WAKEUP_PIN5
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR4 WP1 LL_PWR_SetWakeUpPinPolarityLow • CR4 WP2 LL_PWR_SetWakeUpPinPolarityLow • CR4 WP3 LL_PWR_SetWakeUpPinPolarityLow • CR4 WP4 LL_PWR_SetWakeUpPinPolarityLow

- CR4 WP5 LL_PWR_SetWakeUpPinPolarityLow

LL_PWR_SetWakeUpPinPolarityHigh

Function name	**__STATIC_INLINE void LL_PWR_SetWakeUpPinPolarityHigh (uint32_t WakeUpPin)**
Function description	Set the Wake-Up pin polarity high for the event detection.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: – LL_PWR_WAKEUP_PIN1 – LL_PWR_WAKEUP_PIN2 – LL_PWR_WAKEUP_PIN3 – LL_PWR_WAKEUP_PIN4 – LL_PWR_WAKEUP_PIN5
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR4 WP1 LL_PWR_SetWakeUpPinPolarityHigh • CR4 WP2 LL_PWR_SetWakeUpPinPolarityHigh • CR4 WP3 LL_PWR_SetWakeUpPinPolarityHigh • CR4 WP4 LL_PWR_SetWakeUpPinPolarityHigh • CR4 WP5 LL_PWR_SetWakeUpPinPolarityHigh

LL_PWR_IsWakeUpPinPolarityLow

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsWakeUpPinPolarityLow (uint32_t WakeUpPin)**
Function description	Get the Wake-Up pin polarity for the event detection.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: – LL_PWR_WAKEUP_PIN1 – LL_PWR_WAKEUP_PIN2 – LL_PWR_WAKEUP_PIN3 – LL_PWR_WAKEUP_PIN4 – LL_PWR_WAKEUP_PIN5
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR4 WP1 LL_PWR_IsWakeUpPinPolarityLow • CR4 WP2 LL_PWR_IsWakeUpPinPolarityLow • CR4 WP3 LL_PWR_IsWakeUpPinPolarityLow • CR4 WP4 LL_PWR_IsWakeUpPinPolarityLow • CR4 WP5 LL_PWR_IsWakeUpPinPolarityLow

LL_PWR_EnableGPIOPullUp

Function name	**__STATIC_INLINE void LL_PWR_EnableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)**
Function description	Enable GPIO pull-up state in Standby and Shutdown modes.
Parameters	• **GPIO:** This parameter can be one of the following values: (*) value not defined in all devices – LL_PWR_GPIO_A

- – LL_PWR_GPIO_B
  – LL_PWR_GPIO_C
  – LL_PWR_GPIO_D
  – LL_PWR_GPIO_E
  – LL_PWR_GPIO_F (*)
  – LL_PWR_GPIO_G (*)
  – LL_PWR_GPIO_H
  – LL_PWR_GPIO_I (*)
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values
- **None:**

Reference Manual to LL API cross reference:
- PUCRA PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRB PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRC PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRD PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRE PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRF PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRG PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRH PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRI PU0-11 LL_PWR_EnableGPIOPullUp

LL_PWR_DisableGPIOPullUp

Function name
__STATIC_INLINE void LL_PWR_DisableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description
Disable GPIO pull-up state in Standby and Shutdown modes.

Parameters
- **GPIO:** This parameter can be one of the following values: (*) value not defined in all devices
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D
  - LL_PWR_GPIO_E
  - LL_PWR_GPIO_F (*)
  - LL_PWR_GPIO_G (*)

- LL_PWR_GPIO_H
- LL_PWR_GPIO_I (*)

- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values
- **None:**

Reference Manual to LL API cross reference:
- PUCRA PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRB PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRC PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRD PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRE PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRF PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRG PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRH PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRI PU0-11 LL_PWR_DisableGPIOPullUp

LL_PWR_IsEnabledGPIOPullUp

Function name
__STATIC_INLINE uint32_t LL_PWR_IsEnabledGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description
Check if GPIO pull-up state is enabled.

Parameters
- **GPIO:** This parameter can be one of the following values: (*) value not defined in all devices
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D
  - LL_PWR_GPIO_E
  - LL_PWR_GPIO_F (*)
  - LL_PWR_GPIO_G (*)
  - LL_PWR_GPIO_H
  - LL_PWR_GPIO_I (*)
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1

- LL_PWR_GPIO_BIT_2
- LL_PWR_GPIO_BIT_3
- LL_PWR_GPIO_BIT_4
- LL_PWR_GPIO_BIT_5
- LL_PWR_GPIO_BIT_6
- LL_PWR_GPIO_BIT_7
- LL_PWR_GPIO_BIT_8
- LL_PWR_GPIO_BIT_9
- LL_PWR_GPIO_BIT_10
- LL_PWR_GPIO_BIT_11
- LL_PWR_GPIO_BIT_12
- LL_PWR_GPIO_BIT_13
- LL_PWR_GPIO_BIT_14
- LL_PWR_GPIO_BIT_15

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- PUCRA PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRB PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRC PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRD PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRE PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRF PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRG PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRH PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRI PU0-11 LL_PWR_IsEnabledGPIOPullUp

LL_PWR_EnableGPIOPullDown

Function name
__STATIC_INLINE void LL_PWR_EnableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description
Enable GPIO pull-down state in Standby and Shutdown modes.

Parameters
- **GPIO:** This parameter can be one of the following values: (*) value not defined in all devices
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D
  - LL_PWR_GPIO_E
  - LL_PWR_GPIO_F (*)
  - LL_PWR_GPIO_G (*)
  - LL_PWR_GPIO_H
  - LL_PWR_GPIO_I (*)
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7

- LL_PWR_GPIO_BIT_8
- LL_PWR_GPIO_BIT_9
- LL_PWR_GPIO_BIT_10
- LL_PWR_GPIO_BIT_11
- LL_PWR_GPIO_BIT_12
- LL_PWR_GPIO_BIT_13
- LL_PWR_GPIO_BIT_14
- LL_PWR_GPIO_BIT_15

Return values
- **None:**

Reference Manual to LL API cross reference:
- PDCRA PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRB PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRC PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRD PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRE PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRF PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRG PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRH PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRI PD0-11 LL_PWR_EnableGPIOPullDown

LL_PWR_DisableGPIOPullDown

Function name
__STATIC_INLINE void LL_PWR_DisableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description
Disable GPIO pull-down state in Standby and Shutdown modes.

Parameters
- **GPIO:** This parameter can be one of the following values: (*) value not defined in all devices
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D
  - LL_PWR_GPIO_E
  - LL_PWR_GPIO_F (*)
  - LL_PWR_GPIO_G (*)
  - LL_PWR_GPIO_H
  - LL_PWR_GPIO_I (*)
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13

	– LL_PWR_GPIO_BIT_14 – LL_PWR_GPIO_BIT_15
Return values	• **None:**
Reference Manual to LL API cross reference:	• PDCRA PD0-15 LL_PWR_DisableGPIOPullDown • PDCRB PD0-15 LL_PWR_DisableGPIOPullDown • PDCRC PD0-15 LL_PWR_DisableGPIOPullDown • PDCRD PD0-15 LL_PWR_DisableGPIOPullDown • PDCRE PD0-15 LL_PWR_DisableGPIOPullDown • PDCRF PD0-15 LL_PWR_DisableGPIOPullDown • PDCRG PD0-15 LL_PWR_DisableGPIOPullDown • PDCRH PD0-15 LL_PWR_DisableGPIOPullDown • PDCRI PD0-11 LL_PWR_DisableGPIOPullDown

LL_PWR_IsEnabledGPIOPullDown

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)**
Function description	Check if GPIO pull-down state is enabled.
Parameters	• **GPIO:** This parameter can be one of the following values: (*) value not defined in all devices – LL_PWR_GPIO_A – LL_PWR_GPIO_B – LL_PWR_GPIO_C – LL_PWR_GPIO_D – LL_PWR_GPIO_E – LL_PWR_GPIO_F (*) – LL_PWR_GPIO_G (*) – LL_PWR_GPIO_H – LL_PWR_GPIO_I (*) • **GPIONumber:** This parameter can be one of the following values: – LL_PWR_GPIO_BIT_0 – LL_PWR_GPIO_BIT_1 – LL_PWR_GPIO_BIT_2 – LL_PWR_GPIO_BIT_3 – LL_PWR_GPIO_BIT_4 – LL_PWR_GPIO_BIT_5 – LL_PWR_GPIO_BIT_6 – LL_PWR_GPIO_BIT_7 – LL_PWR_GPIO_BIT_8 – LL_PWR_GPIO_BIT_9 – LL_PWR_GPIO_BIT_10 – LL_PWR_GPIO_BIT_11 – LL_PWR_GPIO_BIT_12 – LL_PWR_GPIO_BIT_13 – LL_PWR_GPIO_BIT_14 – LL_PWR_GPIO_BIT_15
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• PDCRA PD0-15 LL_PWR_IsEnabledGPIOPullDown • PDCRB PD0-15 LL_PWR_IsEnabledGPIOPullDown

reference:	• PDCRC PD0-15 LL_PWR_IsEnabledGPIOPullDown • PDCRD PD0-15 LL_PWR_IsEnabledGPIOPullDown • PDCRE PD0-15 LL_PWR_IsEnabledGPIOPullDown • PDCRF PD0-15 LL_PWR_IsEnabledGPIOPullDown • PDCRG PD0-15 LL_PWR_IsEnabledGPIOPullDown • PDCRH PD0-15 LL_PWR_IsEnabledGPIOPullDown • PDCRI PD0-11 LL_PWR_IsEnabledGPIOPullDown

LL_PWR_IsActiveFlag_InternWU

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_InternWU (void)**
Function description	Get Internal Wake-up line Flag.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR1 WUFI LL_PWR_IsActiveFlag_InternWU

LL_PWR_IsActiveFlag_SB

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_SB (void)**
Function description	Get Stand-By Flag.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR1 SBF LL_PWR_IsActiveFlag_SB

LL_PWR_IsActiveFlag_WU5

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU5 (void)**
Function description	Get Wake-up Flag 5.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR1 WUF5 LL_PWR_IsActiveFlag_WU5

LL_PWR_IsActiveFlag_WU4

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU4 (void)**
Function description	Get Wake-up Flag 4.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR1 WUF4 LL_PWR_IsActiveFlag_WU4

LL_PWR_IsActiveFlag_WU3

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU3 (void)**
Function description	Get Wake-up Flag 3.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR1 WUF3 LL_PWR_IsActiveFlag_WU3

LL_PWR_IsActiveFlag_WU2

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU2 (void)**
Function description	Get Wake-up Flag 2.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR1 WUF2 LL_PWR_IsActiveFlag_WU2

LL_PWR_IsActiveFlag_WU1

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU1 (void)**
Function description	Get Wake-up Flag 1.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR1 WUF1 LL_PWR_IsActiveFlag_WU1

LL_PWR_ClearFlag_SB

Function name	**__STATIC_INLINE void LL_PWR_ClearFlag_SB (void)**
Function description	Clear Stand-By Flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCR CSBF LL_PWR_ClearFlag_SB

LL_PWR_ClearFlag_WU

Function name	**__STATIC_INLINE void LL_PWR_ClearFlag_WU (void)**
Function description	Clear Wake-up Flags.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCR CWUF LL_PWR_ClearFlag_WU

LL_PWR_ClearFlag_WU5

Function name	**__STATIC_INLINE void LL_PWR_ClearFlag_WU5 (void)**
Function description	Clear Wake-up Flag 5.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCR CWUF5 LL_PWR_ClearFlag_WU5

LL_PWR_ClearFlag_WU4

Function name	**__STATIC_INLINE void LL_PWR_ClearFlag_WU4 (void)**
Function description	Clear Wake-up Flag 4.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCR CWUF4 LL_PWR_ClearFlag_WU4

LL_PWR_ClearFlag_WU3

Function name	**__STATIC_INLINE void LL_PWR_ClearFlag_WU3 (void)**
Function description	Clear Wake-up Flag 3.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCR CWUF3 LL_PWR_ClearFlag_WU3

LL_PWR_ClearFlag_WU2

Function name	**__STATIC_INLINE void LL_PWR_ClearFlag_WU2 (void)**
Function description	Clear Wake-up Flag 2.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCR CWUF2 LL_PWR_ClearFlag_WU2

LL_PWR_ClearFlag_WU1

Function name	**__STATIC_INLINE void LL_PWR_ClearFlag_WU1 (void)**
Function description	Clear Wake-up Flag 1.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCR CWUF1 LL_PWR_ClearFlag_WU1

LL_PWR_IsActiveFlag_PVMO4

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVMO4 (void)**
Function description	Indicate whether VDDA voltage is below or above PVM4 threshold.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR2 PVMO4 LL_PWR_IsActiveFlag_PVMO4

LL_PWR_IsActiveFlag_PVMO3

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVMO3 (void)**
Function description	Indicate whether VDDA voltage is below or above PVM3 threshold.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR2 PVMO3 LL_PWR_IsActiveFlag_PVMO3

LL_PWR_IsActiveFlag_PVMO2

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVMO2 (void)**
Function description	Indicate whether VDDIO2 voltage is below or above PVM2 threshold.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR2 PVMO2 LL_PWR_IsActiveFlag_PVMO2

LL_PWR_IsActiveFlag_PVMO1

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVMO1 (void)**
Function description	Indicate whether VDDUSB voltage is below or above PVM1 threshold.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR2 PVMO1 LL_PWR_IsActiveFlag_PVMO1

LL_PWR_IsActiveFlag_PVDO

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVDO (void)**

Function description	Indicate whether VDD voltage is below or above the selected PVD threshold.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR2 PVDO LL_PWR_IsActiveFlag_PVDO

LL_PWR_IsActiveFlag_VOS

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VOS (void)**
Function description	Indicate whether the regulator is ready in the selected voltage range or if its output voltage is still changing to the required voltage level.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR2 VOSF LL_PWR_IsActiveFlag_VOS

LL_PWR_IsActiveFlag_REGLPF

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_REGLPF (void)**
Function description	Indicate whether the regulator is ready in main mode or is in low-power mode.
Return values	• **State:** of bit (1 or 0).
Notes	• Take care, return value "0" means the regulator is ready. Return value "1" means the output voltage range is still changing.
Reference Manual to LL API cross reference:	• SR2 REGLPF LL_PWR_IsActiveFlag_REGLPF

LL_PWR_IsActiveFlag_REGLPS

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_REGLPS (void)**
Function description	Indicate whether or not the low-power regulator is ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR2 REGLPS LL_PWR_IsActiveFlag_REGLPS

LL_PWR_DeInit

Function name	**ErrorStatus LL_PWR_DeInit (void)**
Function description	De-initialize the PWR registers to their default reset values.
Return values	• **An:** ErrorStatus enumeration value:

– SUCCESS: PWR registers are de-initialized
– ERROR: not applicable

90.2 PWR Firmware driver defines

90.2.1 PWR

BATT CHARG RESISTOR

LL_PWR_BATT_CHARG_RESISTOR_5K

LL_PWR_BATT_CHARGRESISTOR_1_5K

Clear Flags Defines

LL_PWR_SCR_CSBF

LL_PWR_SCR_CWUF

LL_PWR_SCR_CWUF5

LL_PWR_SCR_CWUF4

LL_PWR_SCR_CWUF3

LL_PWR_SCR_CWUF2

LL_PWR_SCR_CWUF1

Get Flags Defines

LL_PWR_SR1_WUFI

LL_PWR_SR1_SBF

LL_PWR_SR1_WUF5

LL_PWR_SR1_WUF4

LL_PWR_SR1_WUF3

LL_PWR_SR1_WUF2

LL_PWR_SR1_WUF1

LL_PWR_SR2_PVMO4

LL_PWR_SR2_PVMO3

LL_PWR_SR2_PVMO2

LL_PWR_SR2_PVMO1

LL_PWR_SR2_PVDO

LL_PWR_SR2_VOSF

LL_PWR_SR2_REGLPF

LL_PWR_SR2_REGLPS

GPIO

LL_PWR_GPIO_A

LL_PWR_GPIO_B

LL_PWR_GPIO_C

LL_PWR_GPIO_D

LL_PWR_GPIO_E

LL_PWR_GPIO_F

LL_PWR_GPIO_G

LL_PWR_GPIO_H

LL_PWR_GPIO_I

GPIO BIT

LL_PWR_GPIO_BIT_0

LL_PWR_GPIO_BIT_1

LL_PWR_GPIO_BIT_2

LL_PWR_GPIO_BIT_3

LL_PWR_GPIO_BIT_4

LL_PWR_GPIO_BIT_5

LL_PWR_GPIO_BIT_6

LL_PWR_GPIO_BIT_7

LL_PWR_GPIO_BIT_8

LL_PWR_GPIO_BIT_9

LL_PWR_GPIO_BIT_10

LL_PWR_GPIO_BIT_11

LL_PWR_GPIO_BIT_12

LL_PWR_GPIO_BIT_13

LL_PWR_GPIO_BIT_14

LL_PWR_GPIO_BIT_15

MODE PWR

LL_PWR_MODE_STOP0

LL_PWR_MODE_STOP1

LL_PWR_MODE_STOP2

LL_PWR_MODE_STANDBY

LL_PWR_MODE_SHUTDOWN

PVDLEVEL

LL_PWR_PVDLEVEL_0

LL_PWR_PVDLEVEL_1

LL_PWR_PVDLEVEL_2

LL_PWR_PVDLEVEL_3

LL_PWR_PVDLEVEL_4

LL_PWR_PVDLEVEL_5

LL_PWR_PVDLEVEL_6

LL_PWR_PVDLEVEL_7

Peripheral voltage monitoring

LL_PWR_PVM_VDDUSB_1_2V

LL_PWR_PVM_VDDIO2_0_9V

LL_PWR_PVM_VDDA_1_62V

LL_PWR_PVM_VDDA_2_2V

REGU VOLTAGE

LL_PWR_REGU_VOLTAGE_SCALE1

LL_PWR_REGU_VOLTAGE_SCALE2

WAKEUP

LL_PWR_WAKEUP_PIN1

LL_PWR_WAKEUP_PIN2

LL_PWR_WAKEUP_PIN3

LL_PWR_WAKEUP_PIN4

LL_PWR_WAKEUP_PIN5

Legacy functions name

LL_PWR_IsActiveFlag_VOSF

Common Write and read registers Macros

LL_PWR_WriteReg

Description:

- Write a value in PWR register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_PWR_ReadReg

Description:

- Read a value in PWR register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

91 LL RCC Generic Driver

91.1 RCC Firmware driver registers structures

91.1.1 LL_RCC_ClocksTypeDef

Data Fields

- ***uint32_t SYSCLK_Frequency***
- ***uint32_t HCLK_Frequency***
- ***uint32_t PCLK1_Frequency***
- ***uint32_t PCLK2_Frequency***

Field Documentation

- ***uint32_t LL_RCC_ClocksTypeDef::SYSCLK_Frequency***
  SYSCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::HCLK_Frequency***
  HCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK1_Frequency***
  PCLK1 clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK2_Frequency***
  PCLK2 clock frequency

91.2 RCC Firmware driver API description

91.2.1 Detailed description of functions

LL_RCC_HSE_EnableCSS

Function name	**__STATIC_INLINE void LL_RCC_HSE_EnableCSS (void)**
Function description	Enable the Clock Security System.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CSSON LL_RCC_HSE_EnableCSS

LL_RCC_HSE_EnableBypass

Function name	**__STATIC_INLINE void LL_RCC_HSE_EnableBypass (void)**
Function description	Enable HSE external oscillator (HSE Bypass)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSEBYP LL_RCC_HSE_EnableBypass

LL_RCC_HSE_DisableBypass

Function name	**__STATIC_INLINE void LL_RCC_HSE_DisableBypass (void)**

Function description	Disable HSE external oscillator (HSE Bypass)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSEBYP LL_RCC_HSE_DisableBypass

LL_RCC_HSE_Enable

Function name	**__STATIC_INLINE void LL_RCC_HSE_Enable (void)**
Function description	Enable HSE crystal oscillator (HSE ON)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSEON LL_RCC_HSE_Enable

LL_RCC_HSE_Disable

Function name	**__STATIC_INLINE void LL_RCC_HSE_Disable (void)**
Function description	Disable HSE crystal oscillator (HSE ON)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSEON LL_RCC_HSE_Disable

LL_RCC_HSE_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void)**
Function description	Check if HSE oscillator Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR HSERDY LL_RCC_HSE_IsReady

LL_RCC_HSI_EnableInStopMode

Function name	**__STATIC_INLINE void LL_RCC_HSI_EnableInStopMode (void)**
Function description	Enable HSI even in stop mode.
Return values	• **None:**
Notes	• HSI oscillator is forced ON even in Stop mode
Reference Manual to LL API cross reference:	• CR HSIKERON LL_RCC_HSI_EnableInStopMode

LL_RCC_HSI_DisableInStopMode

Function name	**__STATIC_INLINE void LL_RCC_HSI_DisableInStopMode (void)**
Function description	Disable HSI in stop mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSIKERON LL_RCC_HSI_DisableInStopMode

LL_RCC_HSI_Enable

Function name	**__STATIC_INLINE void LL_RCC_HSI_Enable (void)**
Function description	Enable HSI oscillator.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSION LL_RCC_HSI_Enable

LL_RCC_HSI_Disable

Function name	**__STATIC_INLINE void LL_RCC_HSI_Disable (void)**
Function description	Disable HSI oscillator.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSION LL_RCC_HSI_Disable

LL_RCC_HSI_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI_IsReady (void)**
Function description	Check if HSI clock is ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR HSIRDY LL_RCC_HSI_IsReady

LL_RCC_HSI_EnableAutoFromStop

Function name	**__STATIC_INLINE void LL_RCC_HSI_EnableAutoFromStop (void)**
Function description	Enable HSI Automatic from stop mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSIASFS LL_RCC_HSI_EnableAutoFromStop

LL_RCC_HSI_DisableAutoFromStop

Function name	**__STATIC_INLINE void LL_RCC_HSI_DisableAutoFromStop (void)**
Function description	Disable HSI Automatic from stop mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSIASFS LL_RCC_HSI_DisableAutoFromStop

LL_RCC_HSI_GetCalibration

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibration (void)**
Function description	Get HSI Calibration value.
Return values	• **Between:** Min_Data = 0x00 and Max_Data = 0xFF
Notes	• When HSITRIM is written, HSICAL is updated with the sum of HSITRIM and the factory trim value
Reference Manual to LL API cross reference:	• ICSCR HSICAL LL_RCC_HSI_GetCalibration

LL_RCC_HSI_SetCalibTrimming

Function name	**__STATIC_INLINE void LL_RCC_HSI_SetCalibTrimming (uint32_t Value)**
Function description	Set HSI Calibration trimming.
Parameters	• **Value:** Between Min_Data = 0 and Max_Data = 31
Return values	• **None:**
Notes	• user-programmable trimming value that is added to the HSICAL • Default value is 16, which, when added to the HSICAL value, should trim the HSI to 16 MHz +/- 1 %
Reference Manual to LL API cross reference:	• ICSCR HSITRIM LL_RCC_HSI_SetCalibTrimming

LL_RCC_HSI_GetCalibTrimming

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibTrimming (void)**
Function description	Get HSI Calibration trimming.
Return values	• **Between:** Min_Data = 0 and Max_Data = 31
Reference Manual to LL API cross reference:	• ICSCR HSITRIM LL_RCC_HSI_GetCalibTrimming

LL_RCC_HSI48_Enable

Function name	**__STATIC_INLINE void LL_RCC_HSI48_Enable (void)**
Function description	Enable HSI48.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CRRCR HSI48ON LL_RCC_HSI48_Enable

LL_RCC_HSI48_Disable

Function name	**__STATIC_INLINE void LL_RCC_HSI48_Disable (void)**
Function description	Disable HSI48.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CRRCR HSI48ON LL_RCC_HSI48_Disable

LL_RCC_HSI48_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI48_IsReady (void)**
Function description	Check if HSI48 oscillator Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CRRCR HSI48RDY LL_RCC_HSI48_IsReady

LL_RCC_HSI48_GetCalibration

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI48_GetCalibration (void)**
Function description	Get HSI48 Calibration value.
Return values	• **Between:** Min_Data = 0x00 and Max_Data = 0xFF
Reference Manual to LL API cross reference:	• CRRCR HSI48CAL LL_RCC_HSI48_GetCalibration

LL_RCC_LSE_Enable

Function name	**__STATIC_INLINE void LL_RCC_LSE_Enable (void)**
Function description	Enable Low Speed External (LSE) crystal.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSEON LL_RCC_LSE_Enable

LL_RCC_LSE_Disable

Function name	**__STATIC_INLINE void LL_RCC_LSE_Disable (void)**
Function description	Disable Low Speed External (LSE) crystal.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSEON LL_RCC_LSE_Disable

LL_RCC_LSE_EnableBypass

Function name	**__STATIC_INLINE void LL_RCC_LSE_EnableBypass (void)**
Function description	Enable external clock source (LSE bypass).
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSEBYP LL_RCC_LSE_EnableBypass

LL_RCC_LSE_DisableBypass

Function name	**__STATIC_INLINE void LL_RCC_LSE_DisableBypass (void)**
Function description	Disable external clock source (LSE bypass).
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSEBYP LL_RCC_LSE_DisableBypass

LL_RCC_LSE_SetDriveCapability

Function name	**__STATIC_INLINE void LL_RCC_LSE_SetDriveCapability (uint32_t LSEDrive)**
Function description	Set LSE oscillator drive capability.
Parameters	• **LSEDrive:** This parameter can be one of the following values: – LL_RCC_LSEDRIVE_LOW – LL_RCC_LSEDRIVE_MEDIUMLOW – LL_RCC_LSEDRIVE_MEDIUMHIGH – LL_RCC_LSEDRIVE_HIGH
Return values	• **None:**
Notes	• The oscillator is in Xtal mode when it is not in bypass mode.
Reference Manual to LL API cross reference:	• BDCR LSEDRV LL_RCC_LSE_SetDriveCapability

LL_RCC_LSE_GetDriveCapability

Function name	**__STATIC_INLINE uint32_t LL_RCC_LSE_GetDriveCapability**

	(void)
Function description	Get LSE oscillator drive capability.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_LSEDRIVE_LOW – LL_RCC_LSEDRIVE_MEDIUMLOW – LL_RCC_LSEDRIVE_MEDIUMHIGH – LL_RCC_LSEDRIVE_HIGH
Reference Manual to LL API cross reference:	• BDCR LSEDRV LL_RCC_LSE_GetDriveCapability

LL_RCC_LSE_EnableCSS

Function name	**__STATIC_INLINE void LL_RCC_LSE_EnableCSS (void)**
Function description	Enable Clock security system on LSE.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSECSSON LL_RCC_LSE_EnableCSS

LL_RCC_LSE_DisableCSS

Function name	**__STATIC_INLINE void LL_RCC_LSE_DisableCSS (void)**
Function description	Disable Clock security system on LSE.
Return values	• **None:**
Notes	• Clock security system can be disabled only after a LSE failure detection. In that case it MUST be disabled by software.
Reference Manual to LL API cross reference:	• BDCR LSECSSON LL_RCC_LSE_DisableCSS

LL_RCC_LSE_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_LSE_IsReady (void)**
Function description	Check if LSE oscillator Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• BDCR LSERDY LL_RCC_LSE_IsReady

LL_RCC_LSE_IsCSSDetected

Function name	**__STATIC_INLINE uint32_t LL_RCC_LSE_IsCSSDetected (void)**
Function description	Check if CSS on LSE failure Detection.
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• BDCR LSECSSD LL_RCC_LSE_IsCSSDetected

LL_RCC_LSI_Enable

Function name	**__STATIC_INLINE void LL_RCC_LSI_Enable (void)**
Function description	Enable LSI Oscillator.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR LSION LL_RCC_LSI_Enable

LL_RCC_LSI_Disable

Function name	**__STATIC_INLINE void LL_RCC_LSI_Disable (void)**
Function description	Disable LSI Oscillator.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR LSION LL_RCC_LSI_Disable

LL_RCC_LSI_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_LSI_IsReady (void)**
Function description	Check if LSI is Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR LSIRDY LL_RCC_LSI_IsReady

LL_RCC_MSI_Enable

Function name	**__STATIC_INLINE void LL_RCC_MSI_Enable (void)**
Function description	Enable MSI oscillator.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR MSION LL_RCC_MSI_Enable

LL_RCC_MSI_Disable

Function name	**__STATIC_INLINE void LL_RCC_MSI_Disable (void)**
Function description	Disable MSI oscillator.
Return values	• **None:**
Reference Manual to	• CR MSION LL_RCC_MSI_Disable

LL API cross reference:	

LL_RCC_MSI_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_MSI_IsReady (void)**
Function description	Check if MSI oscillator Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR MSIRDY LL_RCC_MSI_IsReady

LL_RCC_MSI_EnablePLLMode

Function name	**__STATIC_INLINE void LL_RCC_MSI_EnablePLLMode (void)**
Function description	Enable MSI PLL-mode (Hardware auto calibration with LSE)
Return values	• **None:**
Notes	• MSIPLLEN must be enabled after LSE is enabled (LSEON enabled) and ready (LSERDY set by hardware) • hardware protection to avoid enabling MSIPLLEN if LSE is not ready
Reference Manual to LL API cross reference:	• CR MSIPLLEN LL_RCC_MSI_EnablePLLMode

LL_RCC_MSI_DisablePLLMode

Function name	**__STATIC_INLINE void LL_RCC_MSI_DisablePLLMode (void)**
Function description	Disable MSI-PLL mode.
Return values	• **None:**
Notes	• cleared by hardware when LSE is disabled (LSEON = 0) or when the Clock Security System on LSE detects a LSE failure
Reference Manual to LL API cross reference:	• CR MSIPLLEN LL_RCC_MSI_DisablePLLMode

LL_RCC_MSI_EnableRangeSelection

Function name	**__STATIC_INLINE void LL_RCC_MSI_EnableRangeSelection (void)**
Function description	Enable MSI clock range selection with MSIRANGE register.
Return values	• **None:**
Notes	• Write 0 has no effect. After a standby or a reset MSIRGSEL is at 0 and the MSI range value is provided by MSISRANGE
Reference Manual to LL API cross	• CR MSIRGSEL LL_RCC_MSI_EnableRangeSelection

reference:

LL_RCC_MSI_IsEnabledRangeSelect

Function name	**__STATIC_INLINE uint32_t LL_RCC_MSI_IsEnabledRangeSelect (void)**
Function description	Check if MSI clock range is selected with MSIRANGE register.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR MSIRGSEL LL_RCC_MSI_IsEnabledRangeSelect

LL_RCC_MSI_SetRange

Function name	**__STATIC_INLINE void LL_RCC_MSI_SetRange (uint32_t Range)**
Function description	Configure the Internal Multi Speed oscillator (MSI) clock range in run mode.
Parameters	• **Range:** This parameter can be one of the following values: – LL_RCC_MSIRANGE_0 – LL_RCC_MSIRANGE_1 – LL_RCC_MSIRANGE_2 – LL_RCC_MSIRANGE_3 – LL_RCC_MSIRANGE_4 – LL_RCC_MSIRANGE_5 – LL_RCC_MSIRANGE_6 – LL_RCC_MSIRANGE_7 – LL_RCC_MSIRANGE_8 – LL_RCC_MSIRANGE_9 – LL_RCC_MSIRANGE_10 – LL_RCC_MSIRANGE_11
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR MSIRANGE LL_RCC_MSI_SetRange

LL_RCC_MSI_GetRange

Function name	**__STATIC_INLINE uint32_t LL_RCC_MSI_GetRange (void)**
Function description	Get the Internal Multi Speed oscillator (MSI) clock range in run mode.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_MSIRANGE_0 – LL_RCC_MSIRANGE_1 – LL_RCC_MSIRANGE_2 – LL_RCC_MSIRANGE_3 – LL_RCC_MSIRANGE_4 – LL_RCC_MSIRANGE_5 – LL_RCC_MSIRANGE_6

– LL_RCC_MSIRANGE_7
– LL_RCC_MSIRANGE_8
– LL_RCC_MSIRANGE_9
– LL_RCC_MSIRANGE_10
– LL_RCC_MSIRANGE_11

Reference Manual to LL API cross reference:	• CR MSIRANGE LL_RCC_MSI_GetRange

LL_RCC_MSI_SetRangeAfterStandby

Function name	**__STATIC_INLINE void LL_RCC_MSI_SetRangeAfterStandby (uint32_t Range)**
Function description	Configure MSI range used after standby.
Parameters	• **Range:** This parameter can be one of the following values: – LL_RCC_MSISRANGE_4 – LL_RCC_MSISRANGE_5 – LL_RCC_MSISRANGE_6 – LL_RCC_MSISRANGE_7
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR MSISRANGE LL_RCC_MSI_SetRangeAfterStandby

LL_RCC_MSI_GetRangeAfterStandby

Function name	**__STATIC_INLINE uint32_t LL_RCC_MSI_GetRangeAfterStandby (void)**
Function description	Get MSI range used after standby.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_MSISRANGE_4 – LL_RCC_MSISRANGE_5 – LL_RCC_MSISRANGE_6 – LL_RCC_MSISRANGE_7
Reference Manual to LL API cross reference:	• CSR MSISRANGE LL_RCC_MSI_GetRangeAfterStandby

LL_RCC_MSI_GetCalibration

Function name	**__STATIC_INLINE uint32_t LL_RCC_MSI_GetCalibration (void)**
Function description	Get MSI Calibration value.
Return values	• **Between:** Min_Data = 0 and Max_Data = 255
Notes	• When MSITRIM is written, MSICAL is updated with the sum of MSITRIM and the factory trim value
Reference Manual to LL API cross	• ICSCR MSICAL LL_RCC_MSI_GetCalibration

reference:

LL_RCC_MSI_SetCalibTrimming

Function name	**__STATIC_INLINE void LL_RCC_MSI_SetCalibTrimming (uint32_t Value)**
Function description	Set MSI Calibration trimming.
Parameters	• **Value:** Between Min_Data = 0 and Max_Data = 255
Return values	• **None:**
Notes	• user-programmable trimming value that is added to the MSICAL
Reference Manual to LL API cross reference:	• ICSCR MSITRIM LL_RCC_MSI_SetCalibTrimming

LL_RCC_MSI_GetCalibTrimming

Function name	**__STATIC_INLINE uint32_t LL_RCC_MSI_GetCalibTrimming (void)**
Function description	Get MSI Calibration trimming.
Return values	• **Between:** 0 and 255
Reference Manual to LL API cross reference:	• ICSCR MSITRIM LL_RCC_MSI_GetCalibTrimming

LL_RCC_LSCO_Enable

Function name	**__STATIC_INLINE void LL_RCC_LSCO_Enable (void)**
Function description	Enable Low speed clock.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSCOEN LL_RCC_LSCO_Enable

LL_RCC_LSCO_Disable

Function name	**__STATIC_INLINE void LL_RCC_LSCO_Disable (void)**
Function description	Disable Low speed clock.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSCOEN LL_RCC_LSCO_Disable

LL_RCC_LSCO_SetSource

Function name	**__STATIC_INLINE void LL_RCC_LSCO_SetSource (uint32_t**

	Source)
Function description	Configure Low speed clock selection.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_LSCO_CLKSOURCE_LSI – LL_RCC_LSCO_CLKSOURCE_LSE
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSCOSEL LL_RCC_LSCO_SetSource

LL_RCC_LSCO_GetSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_LSCO_GetSource (void)**
Function description	Get Low speed clock selection.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_LSCO_CLKSOURCE_LSI – LL_RCC_LSCO_CLKSOURCE_LSE
Reference Manual to LL API cross reference:	• BDCR LSCOSEL LL_RCC_LSCO_GetSource

LL_RCC_SetSysClkSource

Function name	**__STATIC_INLINE void LL_RCC_SetSysClkSource (uint32_t Source)**
Function description	Configure the system clock source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_SYS_CLKSOURCE_MSI – LL_RCC_SYS_CLKSOURCE_HSI – LL_RCC_SYS_CLKSOURCE_HSE – LL_RCC_SYS_CLKSOURCE_PLL
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR SW LL_RCC_SetSysClkSource

LL_RCC_GetSysClkSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetSysClkSource (void)**
Function description	Get the system clock source.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_SYS_CLKSOURCE_STATUS_MSI – LL_RCC_SYS_CLKSOURCE_STATUS_HSI – LL_RCC_SYS_CLKSOURCE_STATUS_HSE – LL_RCC_SYS_CLKSOURCE_STATUS_PLL
Reference Manual to LL API cross	• CFGR SWS LL_RCC_GetSysClkSource

reference:

LL_RCC_SetAHBPrescaler

Function name	**__STATIC_INLINE void LL_RCC_SetAHBPrescaler (uint32_t Prescaler)**
Function description	Set AHB prescaler.
Parameters	• **Prescaler:** This parameter can be one of the following values: – LL_RCC_SYSCLK_DIV_1 – LL_RCC_SYSCLK_DIV_2 – LL_RCC_SYSCLK_DIV_4 – LL_RCC_SYSCLK_DIV_8 – LL_RCC_SYSCLK_DIV_16 – LL_RCC_SYSCLK_DIV_64 – LL_RCC_SYSCLK_DIV_128 – LL_RCC_SYSCLK_DIV_256 – LL_RCC_SYSCLK_DIV_512
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR HPRE LL_RCC_SetAHBPrescaler

LL_RCC_SetAPB1Prescaler

Function name	**__STATIC_INLINE void LL_RCC_SetAPB1Prescaler (uint32_t Prescaler)**
Function description	Set APB1 prescaler.
Parameters	• **Prescaler:** This parameter can be one of the following values: – LL_RCC_APB1_DIV_1 – LL_RCC_APB1_DIV_2 – LL_RCC_APB1_DIV_4 – LL_RCC_APB1_DIV_8 – LL_RCC_APB1_DIV_16
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR PPRE1 LL_RCC_SetAPB1Prescaler

LL_RCC_SetAPB2Prescaler

Function name	**__STATIC_INLINE void LL_RCC_SetAPB2Prescaler (uint32_t Prescaler)**
Function description	Set APB2 prescaler.
Parameters	• **Prescaler:** This parameter can be one of the following values: – LL_RCC_APB2_DIV_1

- – LL_RCC_APB2_DIV_2
- – LL_RCC_APB2_DIV_4
- – LL_RCC_APB2_DIV_8
- – LL_RCC_APB2_DIV_16

Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR PPRE2 LL_RCC_SetAPB2Prescaler

LL_RCC_GetAHBPrescaler

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetAHBPrescaler (void)**
Function description	Get AHB prescaler.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_SYSCLK_DIV_1 – LL_RCC_SYSCLK_DIV_2 – LL_RCC_SYSCLK_DIV_4 – LL_RCC_SYSCLK_DIV_8 – LL_RCC_SYSCLK_DIV_16 – LL_RCC_SYSCLK_DIV_64 – LL_RCC_SYSCLK_DIV_128 – LL_RCC_SYSCLK_DIV_256 – LL_RCC_SYSCLK_DIV_512
Reference Manual to LL API cross reference:	• CFGR HPRE LL_RCC_GetAHBPrescaler

LL_RCC_GetAPB1Prescaler

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetAPB1Prescaler (void)**
Function description	Get APB1 prescaler.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_APB1_DIV_1 – LL_RCC_APB1_DIV_2 – LL_RCC_APB1_DIV_4 – LL_RCC_APB1_DIV_8 – LL_RCC_APB1_DIV_16
Reference Manual to LL API cross reference:	• CFGR PPRE1 LL_RCC_GetAPB1Prescaler

LL_RCC_GetAPB2Prescaler

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetAPB2Prescaler (void)**
Function description	Get APB2 prescaler.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_APB2_DIV_1 – LL_RCC_APB2_DIV_2

  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Reference Manual to LL API cross reference:
- CFGR PPRE2 LL_RCC_GetAPB2Prescaler

LL_RCC_SetClkAfterWakeFromStop

Function name
: **__STATIC_INLINE void LL_RCC_SetClkAfterWakeFromStop (uint32_t Clock)**

Function description
: Set Clock After Wake-Up From Stop mode.

Parameters
- **Clock:** This parameter can be one of the following values:
  - LL_RCC_STOP_WAKEUPCLOCK_MSI
  - LL_RCC_STOP_WAKEUPCLOCK_HSI

Return values
- **None:**

Reference Manual to LL API cross reference:
- CFGR STOPWUCK LL_RCC_SetClkAfterWakeFromStop

LL_RCC_GetClkAfterWakeFromStop

Function name
: **__STATIC_INLINE uint32_t LL_RCC_GetClkAfterWakeFromStop (void)**

Function description
: Get Clock After Wake-Up From Stop mode.

Return values
- **Returned:** value can be one of the following values:
  - LL_RCC_STOP_WAKEUPCLOCK_MSI
  - LL_RCC_STOP_WAKEUPCLOCK_HSI

Reference Manual to LL API cross reference:
- CFGR STOPWUCK LL_RCC_GetClkAfterWakeFromStop

LL_RCC_ConfigMCO

Function name
: **__STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOxSource, uint32_t MCOxPrescaler)**

Function description
: Configure MCOx.

Parameters
- **MCOxSource:** This parameter can be one of the following values: (*) value not defined in all devices.
  - LL_RCC_MCO1SOURCE_NOCLOCK
  - LL_RCC_MCO1SOURCE_SYSCLK
  - LL_RCC_MCO1SOURCE_MSI
  - LL_RCC_MCO1SOURCE_HSI
  - LL_RCC_MCO1SOURCE_HSE
  - LL_RCC_MCO1SOURCE_HSI48 (*)
  - LL_RCC_MCO1SOURCE_PLLCLK
  - LL_RCC_MCO1SOURCE_LSI
  - LL_RCC_MCO1SOURCE_LSE

	• **MCOxPrescaler:** This parameter can be one of the following values: – LL_RCC_MCO1_DIV_1 – LL_RCC_MCO1_DIV_2 – LL_RCC_MCO1_DIV_4 – LL_RCC_MCO1_DIV_8 – LL_RCC_MCO1_DIV_16
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR MCOSEL LL_RCC_ConfigMCO • CFGR MCOPRE LL_RCC_ConfigMCO

LL_RCC_SetUSARTClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetUSARTClockSource (uint32_t USARTxSource)**
Function description	Configure USARTx clock source.
Parameters	• **USARTxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_USART1_CLKSOURCE_PCLK2 – LL_RCC_USART1_CLKSOURCE_SYSCLK – LL_RCC_USART1_CLKSOURCE_HSI – LL_RCC_USART1_CLKSOURCE_LSE – LL_RCC_USART2_CLKSOURCE_PCLK1 – LL_RCC_USART2_CLKSOURCE_SYSCLK – LL_RCC_USART2_CLKSOURCE_HSI – LL_RCC_USART2_CLKSOURCE_LSE – LL_RCC_USART3_CLKSOURCE_PCLK1 (*) – LL_RCC_USART3_CLKSOURCE_SYSCLK (*) – LL_RCC_USART3_CLKSOURCE_HSI (*) – LL_RCC_USART3_CLKSOURCE_LSE (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR USARTxSEL LL_RCC_SetUSARTClockSource

LL_RCC_SetUARTClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetUARTClockSource (uint32_t UARTxSource)**
Function description	Configure UARTx clock source.
Parameters	• **UARTxSource:** This parameter can be one of the following values: – LL_RCC_UART4_CLKSOURCE_PCLK1 – LL_RCC_UART4_CLKSOURCE_SYSCLK – LL_RCC_UART4_CLKSOURCE_HSI – LL_RCC_UART4_CLKSOURCE_LSE – LL_RCC_UART5_CLKSOURCE_PCLK1 – LL_RCC_UART5_CLKSOURCE_SYSCLK

- – LL_RCC_UART5_CLKSOURCE_HSI
- – LL_RCC_UART5_CLKSOURCE_LSE

Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR UARTxSEL LL_RCC_SetUARTClockSource

LL_RCC_SetLPUARTClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetLPUARTClockSource (uint32_t LPUARTxSource)**
Function description	Configure LPUART1x clock source.
Parameters	• **LPUARTxSource:** This parameter can be one of the following values: – LL_RCC_LPUART1_CLKSOURCE_PCLK1 – LL_RCC_LPUART1_CLKSOURCE_SYSCLK – LL_RCC_LPUART1_CLKSOURCE_HSI – LL_RCC_LPUART1_CLKSOURCE_LSE
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR LPUART1SEL LL_RCC_SetLPUARTClockSource

LL_RCC_SetI2CClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetI2CClockSource (uint32_t I2CxSource)**
Function description	Configure I2Cx clock source.
Parameters	• **I2CxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_I2C1_CLKSOURCE_PCLK1 – LL_RCC_I2C1_CLKSOURCE_SYSCLK – LL_RCC_I2C1_CLKSOURCE_HSI – LL_RCC_I2C2_CLKSOURCE_PCLK1 (*) – LL_RCC_I2C2_CLKSOURCE_SYSCLK (*) – LL_RCC_I2C2_CLKSOURCE_HSI (*) – LL_RCC_I2C3_CLKSOURCE_PCLK1 – LL_RCC_I2C3_CLKSOURCE_SYSCLK – LL_RCC_I2C3_CLKSOURCE_HSI – LL_RCC_I2C4_CLKSOURCE_PCLK1 (*) – LL_RCC_I2C4_CLKSOURCE_SYSCLK (*) – LL_RCC_I2C4_CLKSOURCE_HSI (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR I2CxSEL LL_RCC_SetI2CClockSource

LL_RCC_SetLPTIMClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetLPTIMClockSource (uint32_t LPTIMxSource)**
Function description	Configure LPTIMx clock source.
Parameters	• **LPTIMxSource:** This parameter can be one of the following values: – LL_RCC_LPTIM1_CLKSOURCE_PCLK1 – LL_RCC_LPTIM1_CLKSOURCE_LSI – LL_RCC_LPTIM1_CLKSOURCE_HSI – LL_RCC_LPTIM1_CLKSOURCE_LSE – LL_RCC_LPTIM2_CLKSOURCE_PCLK1 – LL_RCC_LPTIM2_CLKSOURCE_LSI – LL_RCC_LPTIM2_CLKSOURCE_HSI – LL_RCC_LPTIM2_CLKSOURCE_LSE
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR LPTIMxSEL LL_RCC_SetLPTIMClockSource

LL_RCC_SetSAIClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetSAIClockSource (uint32_t SAIxSource)**
Function description	Configure SAIx clock source.
Parameters	• **SAIxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_SAI1_CLKSOURCE_PLLSAI1 – LL_RCC_SAI1_CLKSOURCE_PLLSAI2 (*) – LL_RCC_SAI1_CLKSOURCE_PLL – LL_RCC_SAI1_CLKSOURCE_PIN – LL_RCC_SAI2_CLKSOURCE_PLLSAI1 (*) – LL_RCC_SAI2_CLKSOURCE_PLLSAI2 (*) – LL_RCC_SAI2_CLKSOURCE_PLL (*) – LL_RCC_SAI2_CLKSOURCE_PIN (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR2 SAIxSEL LL_RCC_SetSAIClockSource

LL_RCC_SetSDMMCClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetSDMMCClockSource (uint32_t SDMMCxSource)**
Function description	Configure SDMMC1 clock source.
Parameters	• **SDMMCxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_SDMMC1_CLKSOURCE_NONE (*) – LL_RCC_SDMMC1_CLKSOURCE_HSI48 (*)

- LL_RCC_SDMMC1_CLKSOURCE_PLLSAI1 (*)
- LL_RCC_SDMMC1_CLKSOURCE_PLL
- LL_RCC_SDMMC1_CLKSOURCE_MSI (*)
- LL_RCC_SDMMC1_CLKSOURCE_48CLK (*)

Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR2 SDMMCSEL LL_RCC_SetSDMMCClockSource

LL_RCC_SetRNGClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetRNGClockSource (uint32_t RNGxSource)**
Function description	Configure RNG clock source.
Parameters	• **RNGxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_RNG_CLKSOURCE_NONE (*) – LL_RCC_RNG_CLKSOURCE_HSI48 (*) – LL_RCC_RNG_CLKSOURCE_PLLSAI1 – LL_RCC_RNG_CLKSOURCE_PLL – LL_RCC_RNG_CLKSOURCE_MSI
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR CLK48SEL LL_RCC_SetRNGClockSource

LL_RCC_SetUSBClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetUSBClockSource (uint32_t USBxSource)**
Function description	Configure USB clock source.
Parameters	• **USBxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_USB_CLKSOURCE_NONE (*) – LL_RCC_USB_CLKSOURCE_HSI48 (*) – LL_RCC_USB_CLKSOURCE_PLLSAI1 – LL_RCC_USB_CLKSOURCE_PLL – LL_RCC_USB_CLKSOURCE_MSI
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR CLK48SEL LL_RCC_SetUSBClockSource

LL_RCC_SetADCClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetADCClockSource (uint32_t ADCxSource)**
Function description	Configure ADC clock source.

Parameters	• **ADCxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_ADC_CLKSOURCE_NONE – LL_RCC_ADC_CLKSOURCE_PLLSAI1 – LL_RCC_ADC_CLKSOURCE_PLLSAI2 (*) – LL_RCC_ADC_CLKSOURCE_SYSCLK
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR ADCSEL LL_RCC_SetADCClockSource

LL_RCC_SetDFSDMAudioClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetDFSDMAudioClockSource (uint32_t Source)**
Function description	Configure DFSDM Audio clock source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_DFSDM1_AUDIO_CLKSOURCE_SAI1 – LL_RCC_DFSDM1_AUDIO_CLKSOURCE_HSI – LL_RCC_DFSDM1_AUDIO_CLKSOURCE_MSI
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR2 ADFSDM1SEL LL_RCC_SetDFSDMAudioClockSource

LL_RCC_SetDFSDMClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetDFSDMClockSource (uint32_t DFSDMxSource)**
Function description	Configure DFSDM Kernel clock source.
Parameters	• **DFSDMxSource:** This parameter can be one of the following values: – LL_RCC_DFSDM1_CLKSOURCE_PCLK2 – LL_RCC_DFSDM1_CLKSOURCE_SYSCLK
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR2 DFSDM1SEL LL_RCC_SetDFSDMClockSource

LL_RCC_SetDSIClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetDSIClockSource (uint32_t Source)**
Function description	Configure DSI clock source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_DSI_CLKSOURCE_PHY – LL_RCC_DSI_CLKSOURCE_PLL

Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR2 DSISEL LL_RCC_SetDSIClockSource

LL_RCC_SetLTDCClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetLTDCClockSource (uint32_t Source)**
Function description	Configure LTDC Clock Source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV2 – LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV4 – LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV8 – LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV16
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR2 PLLSAI2DIVR LL_RCC_SetLTDCClockSource

LL_RCC_SetOCTOSPIClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetOCTOSPIClockSource (uint32_t Source)**
Function description	Configure OCTOSPI clock source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_OCTOSPI_CLKSOURCE_SYSCLK – LL_RCC_OCTOSPI_CLKSOURCE_MSI – LL_RCC_OCTOSPI_CLKSOURCE_PLL
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCIPR2 OSPISEL LL_RCC_SetOCTOSPIClockSource

LL_RCC_GetUSARTClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetUSARTClockSource (uint32_t USARTx)**
Function description	Get USARTx clock source.
Parameters	• **USARTx:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_USART1_CLKSOURCE – LL_RCC_USART2_CLKSOURCE – LL_RCC_USART3_CLKSOURCE (*)
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_RCC_USART1_CLKSOURCE_PCLK2 – LL_RCC_USART1_CLKSOURCE_SYSCLK

- LL_RCC_USART1_CLKSOURCE_HSI
- LL_RCC_USART1_CLKSOURCE_LSE
- LL_RCC_USART2_CLKSOURCE_PCLK1
- LL_RCC_USART2_CLKSOURCE_SYSCLK
- LL_RCC_USART2_CLKSOURCE_HSI
- LL_RCC_USART2_CLKSOURCE_LSE
- LL_RCC_USART3_CLKSOURCE_PCLK1 (*)
- LL_RCC_USART3_CLKSOURCE_SYSCLK (*)
- LL_RCC_USART3_CLKSOURCE_HSI (*)
- LL_RCC_USART3_CLKSOURCE_LSE (*)

Reference Manual to LL API cross reference:
- CCIPR USARTxSEL LL_RCC_GetUSARTClockSource

LL_RCC_GetUARTClockSource

Function name
: **__STATIC_INLINE uint32_t LL_RCC_GetUARTClockSource (uint32_t UARTx)**

Function description
: Get UARTx clock source.

Parameters
- **UARTx:** This parameter can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE
  - LL_RCC_UART5_CLKSOURCE

Return values
- **Returned:** value can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE_PCLK1
  - LL_RCC_UART4_CLKSOURCE_SYSCLK
  - LL_RCC_UART4_CLKSOURCE_HSI
  - LL_RCC_UART4_CLKSOURCE_LSE
  - LL_RCC_UART5_CLKSOURCE_PCLK1
  - LL_RCC_UART5_CLKSOURCE_SYSCLK
  - LL_RCC_UART5_CLKSOURCE_HSI
  - LL_RCC_UART5_CLKSOURCE_LSE

Reference Manual to LL API cross reference:
- CCIPR UARTxSEL LL_RCC_GetUARTClockSource

LL_RCC_GetLPUARTClockSource

Function name
: **__STATIC_INLINE uint32_t LL_RCC_GetLPUARTClockSource (uint32_t LPUARTx)**

Function description
: Get LPUARTx clock source.

Parameters
- **LPUARTx:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE

Return values
- **Returned:** value can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE_PCLK1
  - LL_RCC_LPUART1_CLKSOURCE_SYSCLK
  - LL_RCC_LPUART1_CLKSOURCE_HSI
  - LL_RCC_LPUART1_CLKSOURCE_LSE

Reference Manual to LL API cross reference:	• CCIPR LPUART1SEL LL_RCC_GetLPUARTClockSource

LL_RCC_GetI2CClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetI2CClockSource (uint32_t I2Cx)**
Function description	Get I2Cx clock source.
Parameters	• **I2Cx:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_I2C1_CLKSOURCE – LL_RCC_I2C2_CLKSOURCE (*) – LL_RCC_I2C3_CLKSOURCE – LL_RCC_I2C4_CLKSOURCE (*)
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_RCC_I2C1_CLKSOURCE_PCLK1 – LL_RCC_I2C1_CLKSOURCE_SYSCLK – LL_RCC_I2C1_CLKSOURCE_HSI – LL_RCC_I2C2_CLKSOURCE_PCLK1 (*) – LL_RCC_I2C2_CLKSOURCE_SYSCLK (*) – LL_RCC_I2C2_CLKSOURCE_HSI (*) – LL_RCC_I2C3_CLKSOURCE_PCLK1 – LL_RCC_I2C3_CLKSOURCE_SYSCLK – LL_RCC_I2C3_CLKSOURCE_HSI – LL_RCC_I2C4_CLKSOURCE_PCLK1 (*) – LL_RCC_I2C4_CLKSOURCE_SYSCLK (*) – LL_RCC_I2C4_CLKSOURCE_HSI (*)
Reference Manual to LL API cross reference:	• CCIPR I2CxSEL LL_RCC_GetI2CClockSource

LL_RCC_GetLPTIMClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetLPTIMClockSource (uint32_t LPTIMx)**
Function description	Get LPTIMx clock source.
Parameters	• **LPTIMx:** This parameter can be one of the following values: – LL_RCC_LPTIM1_CLKSOURCE – LL_RCC_LPTIM2_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: – LL_RCC_LPTIM1_CLKSOURCE_PCLK1 – LL_RCC_LPTIM1_CLKSOURCE_LSI – LL_RCC_LPTIM1_CLKSOURCE_HSI – LL_RCC_LPTIM1_CLKSOURCE_LSE – LL_RCC_LPTIM2_CLKSOURCE_PCLK1 – LL_RCC_LPTIM2_CLKSOURCE_LSI – LL_RCC_LPTIM2_CLKSOURCE_HSI

– LL_RCC_LPTIM2_CLKSOURCE_LSE

Reference Manual to LL API cross reference:	• CCIPR LPTIMxSEL LL_RCC_GetLPTIMClockSource

LL_RCC_GetSAIClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetSAIClockSource (uint32_t SAIx)**
Function description	Get SAIx clock source.
Parameters	• **SAIx:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_SAI1_CLKSOURCE – LL_RCC_SAI2_CLKSOURCE (*)
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_RCC_SAI1_CLKSOURCE_PLLSAI1 – LL_RCC_SAI1_CLKSOURCE_PLLSAI2 (*) – LL_RCC_SAI1_CLKSOURCE_PLL – LL_RCC_SAI1_CLKSOURCE_PIN – LL_RCC_SAI2_CLKSOURCE_PLLSAI1 (*) – LL_RCC_SAI2_CLKSOURCE_PLLSAI2 (*) – LL_RCC_SAI2_CLKSOURCE_PLL (*) – LL_RCC_SAI2_CLKSOURCE_PIN (*)
Reference Manual to LL API cross reference:	• CCIPR SAIxSEL LL_RCC_GetSAIClockSource

LL_RCC_GetSDMMCClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetSDMMCClockSource (uint32_t SDMMCx)**
Function description	Get SDMMCx clock source.
Parameters	• **SDMMCx:** This parameter can be one of the following values: – LL_RCC_SDMMC1_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_RCC_SDMMC1_CLKSOURCE_NONE (*) – LL_RCC_SDMMC1_CLKSOURCE_HSI48 (*) – LL_RCC_SDMMC1_CLKSOURCE_PLLSAI1 (*) – LL_RCC_SDMMC1_CLKSOURCE_PLL – LL_RCC_SDMMC1_CLKSOURCE_MSI (*) – LL_RCC_SDMMC1_CLKSOURCE_48CLK (*)
Reference Manual to LL API cross reference:	• CCIPR2 SDMMCSEL LL_RCC_GetSDMMCClockSource

LL_RCC_GetRNGClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetRNGClockSource (uint32_t RNGx)**
Function description	Get RNGx clock source.
Parameters	• **RNGx:** This parameter can be one of the following values: – LL_RCC_RNG_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_RCC_RNG_CLKSOURCE_NONE (*) – LL_RCC_RNG_CLKSOURCE_HSI48 (*) – LL_RCC_RNG_CLKSOURCE_PLLSAI1 – LL_RCC_RNG_CLKSOURCE_PLL – LL_RCC_RNG_CLKSOURCE_MSI
Reference Manual to LL API cross reference:	• CCIPR CLK48SEL LL_RCC_GetRNGClockSource

LL_RCC_GetUSBClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetUSBClockSource (uint32_t USBx)**
Function description	Get USBx clock source.
Parameters	• **USBx:** This parameter can be one of the following values: – LL_RCC_USB_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_RCC_USB_CLKSOURCE_NONE (*) – LL_RCC_USB_CLKSOURCE_HSI48 (*) – LL_RCC_USB_CLKSOURCE_PLLSAI1 – LL_RCC_USB_CLKSOURCE_PLL – LL_RCC_USB_CLKSOURCE_MSI
Reference Manual to LL API cross reference:	• CCIPR CLK48SEL LL_RCC_GetUSBClockSource

LL_RCC_GetADCClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetADCClockSource (uint32_t ADCx)**
Function description	Get ADCx clock source.
Parameters	• **ADCx:** This parameter can be one of the following values: – LL_RCC_ADC_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_RCC_ADC_CLKSOURCE_NONE – LL_RCC_ADC_CLKSOURCE_PLLSAI1 – LL_RCC_ADC_CLKSOURCE_PLLSAI2 (*)

	– LL_RCC_ADC_CLKSOURCE_SYSCLK
Reference Manual to LL API cross reference:	• CCIPR ADCSEL LL_RCC_GetADCClockSource

LL_RCC_GetDFSDMAudioClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetDFSDMAudioClockSource (uint32_t DFSDMx)**
Function description	Get DFSDM Audio Clock Source.
Parameters	• **DFSDMx:** This parameter can be one of the following values: – LL_RCC_DFSDM1_AUDIO_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: – LL_RCC_DFSDM1_AUDIO_CLKSOURCE_SAI1 – LL_RCC_DFSDM1_AUDIO_CLKSOURCE_HSI – LL_RCC_DFSDM1_AUDIO_CLKSOURCE_MSI
Reference Manual to LL API cross reference:	• CCIPR2 ADFSDM1SEL LL_RCC_GetDFSDMAudioClockSource

LL_RCC_GetDFSDMClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetDFSDMClockSource (uint32_t DFSDMx)**
Function description	Get DFSDMx Kernel clock source.
Parameters	• **DFSDMx:** This parameter can be one of the following values: – LL_RCC_DFSDM1_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: – LL_RCC_DFSDM1_CLKSOURCE_PCLK2 – LL_RCC_DFSDM1_CLKSOURCE_SYSCLK
Reference Manual to LL API cross reference:	• CCIPR2 DFSDM1SEL LL_RCC_GetDFSDMClockSource

LL_RCC_GetDSIClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetDSIClockSource (uint32_t DSIx)**
Function description	Get DSI Clock Source.
Parameters	• **DSIx:** This parameter can be one of the following values: – LL_RCC_DSI_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: – LL_RCC_DSI_CLKSOURCE_PHY – LL_RCC_DSI_CLKSOURCE_PLL
Reference Manual to LL API cross reference:	• CCIPR2 DSISEL LL_RCC_GetDSIClockSource

LL_RCC_GetLTDCClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetLTDCClockSource (uint32_t LTDCx)**
Function description	Get LTDC Clock Source.
Parameters	• **LTDCx:** This parameter can be one of the following values: – LL_RCC_LTDC_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: – LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV2 – LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV4 – LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV8 – LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV16
Reference Manual to LL API cross reference:	• CCIPR2 PLLSAI2DIVR LL_RCC_GetLTDCClockSource

LL_RCC_GetOCTOSPIClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetOCTOSPIClockSource (uint32_t OCTOSPIx)**
Function description	Get OCTOSPI clock source.
Parameters	• **OCTOSPIx:** This parameter can be one of the following values: – LL_RCC_OCTOSPI_CLKSOURCE
Return values	• **Returned:** value can be one of the following values: – LL_RCC_OCTOSPI_CLKSOURCE_SYSCLK – LL_RCC_OCTOSPI_CLKSOURCE_MSI – LL_RCC_OCTOSPI_CLKSOURCE_PLL
Reference Manual to LL API cross reference:	• CCIPR2 OSPISEL LL_RCC_GetOCTOSPIClockSource

LL_RCC_SetRTCClockSource

Function name	**__STATIC_INLINE void LL_RCC_SetRTCClockSource (uint32_t Source)**
Function description	Set RTC Clock Source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_RTC_CLKSOURCE_NONE – LL_RCC_RTC_CLKSOURCE_LSE – LL_RCC_RTC_CLKSOURCE_LSI – LL_RCC_RTC_CLKSOURCE_HSE_DIV32
Return values	• **None:**
Notes	• Once the RTC clock source has been selected, it cannot be changed anymore unless the Backup domain is reset, or unless a failure is detected on LSE (LSECSSD is set). The BDRST bit can be used to reset them.

Reference Manual to LL API cross reference:	• BDCR RTCSEL LL_RCC_SetRTCClockSource

LL_RCC_GetRTCClockSource

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetRTCClockSource (void)**
Function description	Get RTC Clock Source.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_RTC_CLKSOURCE_NONE – LL_RCC_RTC_CLKSOURCE_LSE – LL_RCC_RTC_CLKSOURCE_LSI – LL_RCC_RTC_CLKSOURCE_HSE_DIV32
Reference Manual to LL API cross reference:	• BDCR RTCSEL LL_RCC_GetRTCClockSource

LL_RCC_EnableRTC

Function name	**__STATIC_INLINE void LL_RCC_EnableRTC (void)**
Function description	Enable RTC.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR RTCEN LL_RCC_EnableRTC

LL_RCC_DisableRTC

Function name	**__STATIC_INLINE void LL_RCC_DisableRTC (void)**
Function description	Disable RTC.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR RTCEN LL_RCC_DisableRTC

LL_RCC_IsEnabledRTC

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledRTC (void)**
Function description	Check if RTC has been enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• BDCR RTCEN LL_RCC_IsEnabledRTC

LL_RCC_ForceBackupDomainReset

Function name	**__STATIC_INLINE void LL_RCC_ForceBackupDomainReset (void)**
Function description	Force the Backup domain reset.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR BDRST LL_RCC_ForceBackupDomainReset

LL_RCC_ReleaseBackupDomainReset

Function name	**__STATIC_INLINE void LL_RCC_ReleaseBackupDomainReset (void)**
Function description	Release the Backup domain reset.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR BDRST LL_RCC_ReleaseBackupDomainReset

LL_RCC_PLL_Enable

Function name	**__STATIC_INLINE void LL_RCC_PLL_Enable (void)**
Function description	Enable PLL.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR PLLON LL_RCC_PLL_Enable

LL_RCC_PLL_Disable

Function name	**__STATIC_INLINE void LL_RCC_PLL_Disable (void)**
Function description	Disable PLL.
Return values	• **None:**
Notes	• Cannot be disabled if the PLL clock is used as the system clock
Reference Manual to LL API cross reference:	• CR PLLON LL_RCC_PLL_Disable

LL_RCC_PLL_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLL_IsReady (void)**
Function description	Check if PLL Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to	• CR PLLRDY LL_RCC_PLL_IsReady

LL API cross reference:

LL_RCC_PLL_ConfigDomain_SYS

Function name	**__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_SYS (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR)**
Function description	Configure PLL used for SYSCLK Domain.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_PLLSOURCE_NONE – LL_RCC_PLLSOURCE_MSI – LL_RCC_PLLSOURCE_HSI – LL_RCC_PLLSOURCE_HSE • **PLLM:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_PLLM_DIV_1 – LL_RCC_PLLM_DIV_2 – LL_RCC_PLLM_DIV_3 – LL_RCC_PLLM_DIV_4 – LL_RCC_PLLM_DIV_5 – LL_RCC_PLLM_DIV_6 – LL_RCC_PLLM_DIV_7 – LL_RCC_PLLM_DIV_8 – LL_RCC_PLLM_DIV_9 (*) – LL_RCC_PLLM_DIV_10 (*) – LL_RCC_PLLM_DIV_11 (*) – LL_RCC_PLLM_DIV_12 (*) – LL_RCC_PLLM_DIV_13 (*) – LL_RCC_PLLM_DIV_14 (*) – LL_RCC_PLLM_DIV_15 (*) – LL_RCC_PLLM_DIV_16 (*) • **PLLN:** Between 8 and 86 • **PLLR:** This parameter can be one of the following values: – LL_RCC_PLLR_DIV_2 – LL_RCC_PLLR_DIV_4 – LL_RCC_PLLR_DIV_6 – LL_RCC_PLLR_DIV_8
Return values	• **None:**
Notes	• PLL Source and PLLM Divider can be written only when PLL, PLLSAI1 and PLLSAI2 (*) are disabled. • PLLN/PLLR can be written only when PLL is disabled.
Reference Manual to LL API cross reference:	• PLLCFGR PLLSRC LL_RCC_PLL_ConfigDomain_SYS • PLLCFGR PLLM LL_RCC_PLL_ConfigDomain_SYS • PLLCFGR PLLN LL_RCC_PLL_ConfigDomain_SYS • PLLCFGR PLLR LL_RCC_PLL_ConfigDomain_SYS

LL_RCC_PLL_ConfigDomain_SAI

Function name	**__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_SAI**

	(uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP)
Function description	Configure PLL used for SAI domain clock.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_PLLSOURCE_NONE – LL_RCC_PLLSOURCE_MSI – LL_RCC_PLLSOURCE_HSI – LL_RCC_PLLSOURCE_HSE • **PLLM:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_PLLM_DIV_1 – LL_RCC_PLLM_DIV_2 – LL_RCC_PLLM_DIV_3 – LL_RCC_PLLM_DIV_4 – LL_RCC_PLLM_DIV_5 – LL_RCC_PLLM_DIV_6 – LL_RCC_PLLM_DIV_7 – LL_RCC_PLLM_DIV_8 – LL_RCC_PLLM_DIV_9 (*) – LL_RCC_PLLM_DIV_10 (*) – LL_RCC_PLLM_DIV_11 (*) – LL_RCC_PLLM_DIV_12 (*) – LL_RCC_PLLM_DIV_13 (*) – LL_RCC_PLLM_DIV_14 (*) – LL_RCC_PLLM_DIV_15 (*) – LL_RCC_PLLM_DIV_16 (*) • **PLLN:** Between 8 and 86 • **PLLP:** This parameter can be one of the following values: – LL_RCC_PLLP_DIV_2 – LL_RCC_PLLP_DIV_3 – LL_RCC_PLLP_DIV_4 – LL_RCC_PLLP_DIV_5 – LL_RCC_PLLP_DIV_6 – LL_RCC_PLLP_DIV_7 – LL_RCC_PLLP_DIV_8 – LL_RCC_PLLP_DIV_9 – LL_RCC_PLLP_DIV_10 – LL_RCC_PLLP_DIV_11 – LL_RCC_PLLP_DIV_12 – LL_RCC_PLLP_DIV_13 – LL_RCC_PLLP_DIV_14 – LL_RCC_PLLP_DIV_15 – LL_RCC_PLLP_DIV_16 – LL_RCC_PLLP_DIV_17 – LL_RCC_PLLP_DIV_18 – LL_RCC_PLLP_DIV_19 – LL_RCC_PLLP_DIV_20 – LL_RCC_PLLP_DIV_21 – LL_RCC_PLLP_DIV_22 – LL_RCC_PLLP_DIV_23 – LL_RCC_PLLP_DIV_24 – LL_RCC_PLLP_DIV_25

- – LL_RCC_PLLP_DIV_26
- – LL_RCC_PLLP_DIV_27
- – LL_RCC_PLLP_DIV_28
- – LL_RCC_PLLP_DIV_29
- – LL_RCC_PLLP_DIV_30
- – LL_RCC_PLLP_DIV_31

Return values
- **None:**

Notes
- PLL Source and PLLM Divider can be written only when PLL, PLLSAI1 and PLLSAI2 (*) are disabled.
- PLLN/PLLP can be written only when PLL is disabled.
- This can be selected for SAI1 or SAI2 (*)

Reference Manual to LL API cross reference:
- PLLCFGR PLLSRC LL_RCC_PLL_ConfigDomain_SAI
- PLLCFGR PLLM LL_RCC_PLL_ConfigDomain_SAI
- PLLCFGR PLLN LL_RCC_PLL_ConfigDomain_SAI
- PLLCFGR PLLPDIV LL_RCC_PLL_ConfigDomain_SAI

LL_RCC_PLL_ConfigDomain_48M

Function name
__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_48M (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLQ)

Function description
Configure PLL used for 48Mhz domain clock.

Parameters
- **Source:** This parameter can be one of the following values:
  - – LL_RCC_PLLSOURCE_NONE
  - – LL_RCC_PLLSOURCE_MSI
  - – LL_RCC_PLLSOURCE_HSI
  - – LL_RCC_PLLSOURCE_HSE
- **PLLM:** This parameter can be one of the following values: (*) value not defined in all devices.
  - – LL_RCC_PLLM_DIV_1
  - – LL_RCC_PLLM_DIV_2
  - – LL_RCC_PLLM_DIV_3
  - – LL_RCC_PLLM_DIV_4
  - – LL_RCC_PLLM_DIV_5
  - – LL_RCC_PLLM_DIV_6
  - – LL_RCC_PLLM_DIV_7
  - – LL_RCC_PLLM_DIV_8
  - – LL_RCC_PLLM_DIV_9 (*)
  - – LL_RCC_PLLM_DIV_10 (*)
  - – LL_RCC_PLLM_DIV_11 (*)
  - – LL_RCC_PLLM_DIV_12 (*)
  - – LL_RCC_PLLM_DIV_13 (*)
  - – LL_RCC_PLLM_DIV_14 (*)
  - – LL_RCC_PLLM_DIV_15 (*)
  - – LL_RCC_PLLM_DIV_16 (*)
- **PLLN:** Between 8 and 86
- **PLLQ:** This parameter can be one of the following values:
  - – LL_RCC_PLLQ_DIV_2
  - – LL_RCC_PLLQ_DIV_4
  - – LL_RCC_PLLQ_DIV_6

– LL_RCC_PLLQ_DIV_8

Return values	• **None:**
Notes	• PLL Source and PLLM Divider can be written only when PLL, PLLSAI1 and PLLSAI2 (*) are disabled. • PLLN/PLLQ can be written only when PLL is disabled. • This can be selected for USB, RNG, SDMMC
Reference Manual to LL API cross reference:	• PLLCFGR PLLSRC LL_RCC_PLL_ConfigDomain_48M • PLLCFGR PLLM LL_RCC_PLL_ConfigDomain_48M • PLLCFGR PLLN LL_RCC_PLL_ConfigDomain_48M • PLLCFGR PLLQ LL_RCC_PLL_ConfigDomain_48M

LL_RCC_PLL_GetN

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLL_GetN (void)**
Function description	Get Main PLL multiplication factor for VCO.
Return values	• **Between:** 8 and 86
Reference Manual to LL API cross reference:	• PLLCFGR PLLN LL_RCC_PLL_GetN

LL_RCC_PLL_GetP

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLL_GetP (void)**
Function description	Get Main PLL division factor for PLLP.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLLP_DIV_2 – LL_RCC_PLLP_DIV_3 – LL_RCC_PLLP_DIV_4 – LL_RCC_PLLP_DIV_5 – LL_RCC_PLLP_DIV_6 – LL_RCC_PLLP_DIV_7 – LL_RCC_PLLP_DIV_8 – LL_RCC_PLLP_DIV_9 – LL_RCC_PLLP_DIV_10 – LL_RCC_PLLP_DIV_11 – LL_RCC_PLLP_DIV_12 – LL_RCC_PLLP_DIV_13 – LL_RCC_PLLP_DIV_14 – LL_RCC_PLLP_DIV_15 – LL_RCC_PLLP_DIV_16 – LL_RCC_PLLP_DIV_17 – LL_RCC_PLLP_DIV_18 – LL_RCC_PLLP_DIV_19 – LL_RCC_PLLP_DIV_20 – LL_RCC_PLLP_DIV_21 – LL_RCC_PLLP_DIV_22 – LL_RCC_PLLP_DIV_23 – LL_RCC_PLLP_DIV_24 – LL_RCC_PLLP_DIV_25

– LL_RCC_PLLP_DIV_26
– LL_RCC_PLLP_DIV_27
– LL_RCC_PLLP_DIV_28
– LL_RCC_PLLP_DIV_29
– LL_RCC_PLLP_DIV_30
– LL_RCC_PLLP_DIV_31

Notes
- Used for PLLSAI3CLK (SAI1 and SAI2 clock)

Reference Manual to LL API cross reference:
- PLLCFGR PLLPDIV LL_RCC_PLL_GetP

LL_RCC_PLL_GetQ

Function name
__STATIC_INLINE uint32_t LL_RCC_PLL_GetQ (void)

Function description
Get Main PLL division factor for PLLQ.

Return values
- **Returned:** value can be one of the following values:
  – LL_RCC_PLLQ_DIV_2
  – LL_RCC_PLLQ_DIV_4
  – LL_RCC_PLLQ_DIV_6
  – LL_RCC_PLLQ_DIV_8

Notes
- Used for PLL48M1CLK selected for USB, RNG, SDMMC (48 MHz clock)

Reference Manual to LL API cross reference:
- PLLCFGR PLLQ LL_RCC_PLL_GetQ

LL_RCC_PLL_GetR

Function name
__STATIC_INLINE uint32_t LL_RCC_PLL_GetR (void)

Function description
Get Main PLL division factor for PLLR.

Return values
- **Returned:** value can be one of the following values:
  – LL_RCC_PLLR_DIV_2
  – LL_RCC_PLLR_DIV_4
  – LL_RCC_PLLR_DIV_6
  – LL_RCC_PLLR_DIV_8

Notes
- Used for PLLCLK (system clock)

Reference Manual to LL API cross reference:
- PLLCFGR PLLR LL_RCC_PLL_GetR

LL_RCC_PLL_GetMainSource

Function name
__STATIC_INLINE uint32_t LL_RCC_PLL_GetMainSource (void)

Function description
Get the oscillator used as PLL clock source.

Return values
- **Returned:** value can be one of the following values:
  – LL_RCC_PLLSOURCE_NONE

- LL_RCC_PLLSOURCE_MSI
- LL_RCC_PLLSOURCE_HSI
- LL_RCC_PLLSOURCE_HSE

Reference Manual to LL API cross reference:	• PLLCFGR PLLSRC LL_RCC_PLL_GetMainSource

LL_RCC_PLL_GetDivider

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLL_GetDivider (void)**
Function description	Get Division factor for the main PLL and other PLL.
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_RCC_PLLM_DIV_1 – LL_RCC_PLLM_DIV_2 – LL_RCC_PLLM_DIV_3 – LL_RCC_PLLM_DIV_4 – LL_RCC_PLLM_DIV_5 – LL_RCC_PLLM_DIV_6 – LL_RCC_PLLM_DIV_7 – LL_RCC_PLLM_DIV_8 – LL_RCC_PLLM_DIV_9 (*) – LL_RCC_PLLM_DIV_10 (*) – LL_RCC_PLLM_DIV_11 (*) – LL_RCC_PLLM_DIV_12 (*) – LL_RCC_PLLM_DIV_13 (*) – LL_RCC_PLLM_DIV_14 (*) – LL_RCC_PLLM_DIV_15 (*) – LL_RCC_PLLM_DIV_16 (*)
Reference Manual to LL API cross reference:	• PLLCFGR PLLM LL_RCC_PLL_GetDivider

LL_RCC_PLL_EnableDomain_SAI

Function name	**__STATIC_INLINE void LL_RCC_PLL_EnableDomain_SAI (void)**
Function description	Enable PLL output mapped on SAI domain clock.
Return values	• **None:**
Reference Manual to LL API cross reference:	• PLLCFGR PLLPEN LL_RCC_PLL_EnableDomain_SAI

LL_RCC_PLL_DisableDomain_SAI

Function name	**__STATIC_INLINE void LL_RCC_PLL_DisableDomain_SAI (void)**
Function description	Disable PLL output mapped on SAI domain clock.

Return values	• **None:**
Notes	• Cannot be disabled if the PLL clock is used as the system clock • In order to save power, when the PLLCLK of the PLL is not used, should be 0
Reference Manual to LL API cross reference:	• PLLCFGR PLLPEN LL_RCC_PLL_DisableDomain_SAI

LL_RCC_PLL_EnableDomain_48M

Function name	**__STATIC_INLINE void LL_RCC_PLL_EnableDomain_48M (void)**
Function description	Enable PLL output mapped on 48MHz domain clock.
Return values	• **None:**
Reference Manual to LL API cross reference:	• PLLCFGR PLLQEN LL_RCC_PLL_EnableDomain_48M

LL_RCC_PLL_DisableDomain_48M

Function name	**__STATIC_INLINE void LL_RCC_PLL_DisableDomain_48M (void)**
Function description	Disable PLL output mapped on 48MHz domain clock.
Return values	• **None:**
Notes	• Cannot be disabled if the PLL clock is used as the system clock • In order to save power, when the PLLCLK of the PLL is not used, should be 0
Reference Manual to LL API cross reference:	• PLLCFGR PLLQEN LL_RCC_PLL_DisableDomain_48M

LL_RCC_PLL_EnableDomain_SYS

Function name	**__STATIC_INLINE void LL_RCC_PLL_EnableDomain_SYS (void)**
Function description	Enable PLL output mapped on SYSCLK domain.
Return values	• **None:**
Reference Manual to LL API cross reference:	• PLLCFGR PLLREN LL_RCC_PLL_EnableDomain_SYS

LL_RCC_PLL_DisableDomain_SYS

Function name	**__STATIC_INLINE void LL_RCC_PLL_DisableDomain_SYS (void)**

Function description	Disable PLL output mapped on SYSCLK domain.
Return values	• **None:**
Notes	• Cannot be disabled if the PLL clock is used as the system clock • In order to save power, when the PLLCLK of the PLL is not used, Main PLL should be 0
Reference Manual to LL API cross reference:	• PLLCFGR PLLREN LL_RCC_PLL_DisableDomain_SYS

LL_RCC_PLLSAI1_Enable

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_Enable (void)**
Function description	Enable PLLSAI1.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR PLLSAI1ON LL_RCC_PLLSAI1_Enable

LL_RCC_PLLSAI1_Disable

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_Disable (void)**
Function description	Disable PLLSAI1.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR PLLSAI1ON LL_RCC_PLLSAI1_Disable

LL_RCC_PLLSAI1_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI1_IsReady (void)**
Function description	Check if PLLSAI1 Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR PLLSAI1RDY LL_RCC_PLLSAI1_IsReady

LL_RCC_PLLSAI1_ConfigDomain_48M

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_ConfigDomain_48M (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLQ)**
Function description	Configure PLLSAI1 used for 48Mhz domain clock.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_PLLSOURCE_NONE – LL_RCC_PLLSOURCE_MSI – LL_RCC_PLLSOURCE_HSI

	– LL_RCC_PLLSOURCE_HSE • **PLLM:** This parameter can be one of the following values: – LL_RCC_PLLSAI1M_DIV_1 – LL_RCC_PLLSAI1M_DIV_2 – LL_RCC_PLLSAI1M_DIV_3 – LL_RCC_PLLSAI1M_DIV_4 – LL_RCC_PLLSAI1M_DIV_5 – LL_RCC_PLLSAI1M_DIV_6 – LL_RCC_PLLSAI1M_DIV_7 – LL_RCC_PLLSAI1M_DIV_8 – LL_RCC_PLLSAI1M_DIV_9 – LL_RCC_PLLSAI1M_DIV_10 – LL_RCC_PLLSAI1M_DIV_11 – LL_RCC_PLLSAI1M_DIV_12 – LL_RCC_PLLSAI1M_DIV_13 – LL_RCC_PLLSAI1M_DIV_14 – LL_RCC_PLLSAI1M_DIV_15 – LL_RCC_PLLSAI1M_DIV_16 • **PLLN:** Between 8 and 86 • **PLLQ:** This parameter can be one of the following values: – LL_RCC_PLLSAI1Q_DIV_2 – LL_RCC_PLLSAI1Q_DIV_4 – LL_RCC_PLLSAI1Q_DIV_6 – LL_RCC_PLLSAI1Q_DIV_8
Return values	• **None:**
Notes	• PLL Source can be written only when PLL, PLLSAI1 and PLLSAI2 (*) are disabled. • PLLSAI1M/PLLSAI1N/PLLSAI1Q can be written only when PLLSAI1 is disabled. • This can be selected for USB, RNG, SDMMC
Reference Manual to LL API cross reference:	• PLLCFGR PLLSRC LL_RCC_PLLSAI1_ConfigDomain_48M • PLLSAI1CFGR PLLSAI1M LL_RCC_PLLSAI1_ConfigDomain_48M • PLLSAI1CFGR PLLSAI1N LL_RCC_PLLSAI1_ConfigDomain_48M • PLLSAI1CFGR PLLSAI1Q LL_RCC_PLLSAI1_ConfigDomain_48M

LL_RCC_PLLSAI1_ConfigDomain_SAI

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_ConfigDomain_SAI (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP)**
Function description	Configure PLLSAI1 used for SAI domain clock.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_PLLSOURCE_NONE – LL_RCC_PLLSOURCE_MSI – LL_RCC_PLLSOURCE_HSI – LL_RCC_PLLSOURCE_HSE • **PLLM:** This parameter can be one of the following values:

- LL_RCC_PLLSAI1M_DIV_1
- LL_RCC_PLLSAI1M_DIV_2
- LL_RCC_PLLSAI1M_DIV_3
- LL_RCC_PLLSAI1M_DIV_4
- LL_RCC_PLLSAI1M_DIV_5
- LL_RCC_PLLSAI1M_DIV_6
- LL_RCC_PLLSAI1M_DIV_7
- LL_RCC_PLLSAI1M_DIV_8
- LL_RCC_PLLSAI1M_DIV_9
- LL_RCC_PLLSAI1M_DIV_10
- LL_RCC_PLLSAI1M_DIV_11
- LL_RCC_PLLSAI1M_DIV_12
- LL_RCC_PLLSAI1M_DIV_13
- LL_RCC_PLLSAI1M_DIV_14
- LL_RCC_PLLSAI1M_DIV_15
- LL_RCC_PLLSAI1M_DIV_16

• **PLLN:** Between 8 and 86

• **PLLP:** This parameter can be one of the following values:
- LL_RCC_PLLSAI1P_DIV_2
- LL_RCC_PLLSAI1P_DIV_3
- LL_RCC_PLLSAI1P_DIV_4
- LL_RCC_PLLSAI1P_DIV_5
- LL_RCC_PLLSAI1P_DIV_6
- LL_RCC_PLLSAI1P_DIV_7
- LL_RCC_PLLSAI1P_DIV_8
- LL_RCC_PLLSAI1P_DIV_9
- LL_RCC_PLLSAI1P_DIV_10
- LL_RCC_PLLSAI1P_DIV_11
- LL_RCC_PLLSAI1P_DIV_12
- LL_RCC_PLLSAI1P_DIV_13
- LL_RCC_PLLSAI1P_DIV_14
- LL_RCC_PLLSAI1P_DIV_15
- LL_RCC_PLLSAI1P_DIV_16
- LL_RCC_PLLSAI1P_DIV_17
- LL_RCC_PLLSAI1P_DIV_18
- LL_RCC_PLLSAI1P_DIV_19
- LL_RCC_PLLSAI1P_DIV_20
- LL_RCC_PLLSAI1P_DIV_21
- LL_RCC_PLLSAI1P_DIV_22
- LL_RCC_PLLSAI1P_DIV_23
- LL_RCC_PLLSAI1P_DIV_24
- LL_RCC_PLLSAI1P_DIV_25
- LL_RCC_PLLSAI1P_DIV_26
- LL_RCC_PLLSAI1P_DIV_27
- LL_RCC_PLLSAI1P_DIV_28
- LL_RCC_PLLSAI1P_DIV_29
- LL_RCC_PLLSAI1P_DIV_30
- LL_RCC_PLLSAI1P_DIV_31

Return values
• **None:**

Notes
• PLL Source can be written only when PLL, PLLSAI1 and PLLSAI2 (*) are disabled.

	• PLLSAI1M/PLLSAI1N/PLLSAI1PDIV can be written only when PLLSAI1 is disabled. • This can be selected for SAI1 or SAI2
Reference Manual to LL API cross reference:	• PLLCFGR PLLSRC LL_RCC_PLLSAI1_ConfigDomain_SAI • PLLSAI1CFGR PLLSAI1M LL_RCC_PLLSAI1_ConfigDomain_SAI • PLLSAI1CFGR PLLSAI1N LL_RCC_PLLSAI1_ConfigDomain_SAI • PLLSAI1CFGR PLLSAI1PDIV LL_RCC_PLLSAI1_ConfigDomain_SAI

LL_RCC_PLLSAI1_ConfigDomain_ADC

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_ConfigDomain_ADC (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR)**
Function description	Configure PLLSAI1 used for ADC domain clock.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_PLLSOURCE_NONE – LL_RCC_PLLSOURCE_MSI – LL_RCC_PLLSOURCE_HSI – LL_RCC_PLLSOURCE_HSE • **PLLM:** This parameter can be one of the following values: – LL_RCC_PLLSAI1M_DIV_1 – LL_RCC_PLLSAI1M_DIV_2 – LL_RCC_PLLSAI1M_DIV_3 – LL_RCC_PLLSAI1M_DIV_4 – LL_RCC_PLLSAI1M_DIV_5 – LL_RCC_PLLSAI1M_DIV_6 – LL_RCC_PLLSAI1M_DIV_7 – LL_RCC_PLLSAI1M_DIV_8 – LL_RCC_PLLSAI1M_DIV_9 – LL_RCC_PLLSAI1M_DIV_10 – LL_RCC_PLLSAI1M_DIV_11 – LL_RCC_PLLSAI1M_DIV_12 – LL_RCC_PLLSAI1M_DIV_13 – LL_RCC_PLLSAI1M_DIV_14 – LL_RCC_PLLSAI1M_DIV_15 – LL_RCC_PLLSAI1M_DIV_16 • **PLLN:** Between 8 and 86 • **PLLR:** This parameter can be one of the following values: – LL_RCC_PLLSAI1R_DIV_2 – LL_RCC_PLLSAI1R_DIV_4 – LL_RCC_PLLSAI1R_DIV_6 – LL_RCC_PLLSAI1R_DIV_8
Return values	• **None:**
Notes	• PLL Source can be written only when PLL, PLLSAI1 and PLLSAI2 (*) are disabled. • PLLSAI1M/PLLSAI1N/PLLSAI1R can be written only when PLLSAI1 is disabled.

	• This can be selected for ADC
Reference Manual to LL API cross reference:	• PLLCFGR PLLSRC LL_RCC_PLLSAI1_ConfigDomain_ADC • PLLSAI1CFGR PLLSAI1M LL_RCC_PLLSAI1_ConfigDomain_ADC • PLLSAI1CFGR PLLSAI1N LL_RCC_PLLSAI1_ConfigDomain_ADC • PLLSAI1CFGR PLLSAI1R LL_RCC_PLLSAI1_ConfigDomain_ADC

LL_RCC_PLLSAI1_GetN

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI1_GetN (void)**
Function description	Get SAI1PLL multiplication factor for VCO.
Return values	• **Between:** 8 and 86
Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1N LL_RCC_PLLSAI1_GetN

LL_RCC_PLLSAI1_GetP

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI1_GetP (void)**
Function description	Get SAI1PLL division factor for PLLSAI1P.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLLSAI1P_DIV_2 – LL_RCC_PLLSAI1P_DIV_3 – LL_RCC_PLLSAI1P_DIV_4 – LL_RCC_PLLSAI1P_DIV_5 – LL_RCC_PLLSAI1P_DIV_6 – LL_RCC_PLLSAI1P_DIV_7 – LL_RCC_PLLSAI1P_DIV_8 – LL_RCC_PLLSAI1P_DIV_9 – LL_RCC_PLLSAI1P_DIV_10 – LL_RCC_PLLSAI1P_DIV_11 – LL_RCC_PLLSAI1P_DIV_12 – LL_RCC_PLLSAI1P_DIV_13 – LL_RCC_PLLSAI1P_DIV_14 – LL_RCC_PLLSAI1P_DIV_15 – LL_RCC_PLLSAI1P_DIV_16 – LL_RCC_PLLSAI1P_DIV_17 – LL_RCC_PLLSAI1P_DIV_18 – LL_RCC_PLLSAI1P_DIV_19 – LL_RCC_PLLSAI1P_DIV_20 – LL_RCC_PLLSAI1P_DIV_21 – LL_RCC_PLLSAI1P_DIV_22 – LL_RCC_PLLSAI1P_DIV_23 – LL_RCC_PLLSAI1P_DIV_24 – LL_RCC_PLLSAI1P_DIV_25 – LL_RCC_PLLSAI1P_DIV_26 – LL_RCC_PLLSAI1P_DIV_27 – LL_RCC_PLLSAI1P_DIV_28

	– LL_RCC_PLLSAI1P_DIV_29 – LL_RCC_PLLSAI1P_DIV_30 – LL_RCC_PLLSAI1P_DIV_31
Notes	• Used for PLLSAI1CLK (SAI1 or SAI2 (*) clock).
Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1PDIV LL_RCC_PLLSAI1_GetP

LL_RCC_PLLSAI1_GetQ

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI1_GetQ (void)**
Function description	Get SAI1PLL division factor for PLLSAI1Q.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLLSAI1Q_DIV_2 – LL_RCC_PLLSAI1Q_DIV_4 – LL_RCC_PLLSAI1Q_DIV_6 – LL_RCC_PLLSAI1Q_DIV_8
Notes	• Used PLL48M2CLK selected for USB, RNG, SDMMC (48 MHz clock)
Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1Q LL_RCC_PLLSAI1_GetQ

LL_RCC_PLLSAI1_GetR

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI1_GetR (void)**
Function description	Get PLLSAI1 division factor for PLLSAIR.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLLSAI1R_DIV_2 – LL_RCC_PLLSAI1R_DIV_4 – LL_RCC_PLLSAI1R_DIV_6 – LL_RCC_PLLSAI1R_DIV_8
Notes	• Used for PLLADC1CLK (ADC clock)
Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1R LL_RCC_PLLSAI1_GetR

LL_RCC_PLLSAI1_GetDivider

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI1_GetDivider (void)**
Function description	Get Division factor for the PLLSAI1.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLLSAI1M_DIV_1 – LL_RCC_PLLSAI1M_DIV_2 – LL_RCC_PLLSAI1M_DIV_3 – LL_RCC_PLLSAI1M_DIV_4

- LL_RCC_PLLSAI1M_DIV_5
- LL_RCC_PLLSAI1M_DIV_6
- LL_RCC_PLLSAI1M_DIV_7
- LL_RCC_PLLSAI1M_DIV_8
- LL_RCC_PLLSAI1M_DIV_9
- LL_RCC_PLLSAI1M_DIV_10
- LL_RCC_PLLSAI1M_DIV_11
- LL_RCC_PLLSAI1M_DIV_12
- LL_RCC_PLLSAI1M_DIV_13
- LL_RCC_PLLSAI1M_DIV_14
- LL_RCC_PLLSAI1M_DIV_15
- LL_RCC_PLLSAI1M_DIV_16

Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1M LL_RCC_PLLSAI1_GetDivider

LL_RCC_PLLSAI1_EnableDomain_SAI

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_EnableDomain_SAI (void)**
Function description	Enable PLLSAI1 output mapped on SAI domain clock.
Return values	• **None:**
Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1PEN LL_RCC_PLLSAI1_EnableDomain_SAI

LL_RCC_PLLSAI1_DisableDomain_SAI

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_DisableDomain_SAI (void)**
Function description	Disable PLLSAI1 output mapped on SAI domain clock.
Return values	• **None:**
Notes	• In order to save power, when of the PLLSAI1 is not used, should be 0
Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1PEN LL_RCC_PLLSAI1_DisableDomain_SAI

LL_RCC_PLLSAI1_EnableDomain_48M

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_EnableDomain_48M (void)**
Function description	Enable PLLSAI1 output mapped on 48MHz domain clock.
Return values	• **None:**
Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1QEN LL_RCC_PLLSAI1_EnableDomain_48M

LL_RCC_PLLSAI1_DisableDomain_48M

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_DisableDomain_48M (void)**
Function description	Disable PLLSAI1 output mapped on 48MHz domain clock.
Return values	• **None:**
Notes	• In order to save power, when of the PLLSAI1 is not used, should be 0
Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1QEN LL_RCC_PLLSAI1_DisableDomain_48M

LL_RCC_PLLSAI1_EnableDomain_ADC

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_EnableDomain_ADC (void)**
Function description	Enable PLLSAI1 output mapped on ADC domain clock.
Return values	• **None:**
Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1REN LL_RCC_PLLSAI1_EnableDomain_ADC

LL_RCC_PLLSAI1_DisableDomain_ADC

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI1_DisableDomain_ADC (void)**
Function description	Disable PLLSAI1 output mapped on ADC domain clock.
Return values	• **None:**
Notes	• In order to save power, when of the PLLSAI1 is not used, Main PLLSAI1 should be 0
Reference Manual to LL API cross reference:	• PLLSAI1CFGR PLLSAI1REN LL_RCC_PLLSAI1_DisableDomain_ADC

LL_RCC_PLLSAI2_Enable

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI2_Enable (void)**
Function description	Enable PLLSAI2.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR PLLSAI2ON LL_RCC_PLLSAI2_Enable

LL_RCC_PLLSAI2_Disable

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI2_Disable (void)**

Function description	Disable PLLSAI2.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR PLLSAI2ON LL_RCC_PLLSAI2_Disable

LL_RCC_PLLSAI2_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI2_IsReady (void)**
Function description	Check if PLLSAI2 Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR PLLSAI2RDY LL_RCC_PLLSAI2_IsReady

LL_RCC_PLLSAI2_ConfigDomain_SAI

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI2_ConfigDomain_SAI (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP)**
Function description	Configure PLLSAI2 used for SAI domain clock.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_PLLSOURCE_NONE – LL_RCC_PLLSOURCE_MSI – LL_RCC_PLLSOURCE_HSI – LL_RCC_PLLSOURCE_HSE • **PLLM:** This parameter can be one of the following values: – LL_RCC_PLLSAI2M_DIV_1 – LL_RCC_PLLSAI2M_DIV_2 – LL_RCC_PLLSAI2M_DIV_3 – LL_RCC_PLLSAI2M_DIV_4 – LL_RCC_PLLSAI2M_DIV_5 – LL_RCC_PLLSAI2M_DIV_6 – LL_RCC_PLLSAI2M_DIV_7 – LL_RCC_PLLSAI2M_DIV_8 – LL_RCC_PLLSAI2M_DIV_9 – LL_RCC_PLLSAI2M_DIV_10 – LL_RCC_PLLSAI2M_DIV_11 – LL_RCC_PLLSAI2M_DIV_12 – LL_RCC_PLLSAI2M_DIV_13 – LL_RCC_PLLSAI2M_DIV_14 – LL_RCC_PLLSAI2M_DIV_15 – LL_RCC_PLLSAI2M_DIV_16 • **PLLN:** Between 8 and 86 • **PLLP:** This parameter can be one of the following values: – LL_RCC_PLLSAI2P_DIV_2 – LL_RCC_PLLSAI2P_DIV_3 – LL_RCC_PLLSAI2P_DIV_4 – LL_RCC_PLLSAI2P_DIV_5 – LL_RCC_PLLSAI2P_DIV_6

– LL_RCC_PLLSAI2P_DIV_7
– LL_RCC_PLLSAI2P_DIV_8
– LL_RCC_PLLSAI2P_DIV_9
– LL_RCC_PLLSAI2P_DIV_10
– LL_RCC_PLLSAI2P_DIV_11
– LL_RCC_PLLSAI2P_DIV_12
– LL_RCC_PLLSAI2P_DIV_13
– LL_RCC_PLLSAI2P_DIV_14
– LL_RCC_PLLSAI2P_DIV_15
– LL_RCC_PLLSAI2P_DIV_16
– LL_RCC_PLLSAI2P_DIV_17
– LL_RCC_PLLSAI2P_DIV_18
– LL_RCC_PLLSAI2P_DIV_19
– LL_RCC_PLLSAI2P_DIV_20
– LL_RCC_PLLSAI2P_DIV_21
– LL_RCC_PLLSAI2P_DIV_22
– LL_RCC_PLLSAI2P_DIV_23
– LL_RCC_PLLSAI2P_DIV_24
– LL_RCC_PLLSAI2P_DIV_25
– LL_RCC_PLLSAI2P_DIV_26
– LL_RCC_PLLSAI2P_DIV_27
– LL_RCC_PLLSAI2P_DIV_28
– LL_RCC_PLLSAI2P_DIV_29
– LL_RCC_PLLSAI2P_DIV_30
– LL_RCC_PLLSAI2P_DIV_31

Return values
- **None:**

Notes
- PLL Source can be written only when PLL, PLLSAI1 and PLLSAI2 (*) are disabled.
- PLLSAI2M/PLLSAI2N/PLLSAI2PDIV can be written only when PLLSAI2 is disabled.
- This can be selected for SAI1 or SAI2

Reference Manual to LL API cross reference:
- PLLCFGR PLLSRC LL_RCC_PLLSAI2_ConfigDomain_SAI
- PLLSAI2CFGR PLLSAI2M LL_RCC_PLLSAI2_ConfigDomain_SAI
- PLLSAI2CFGR PLLSAI2N LL_RCC_PLLSAI2_ConfigDomain_SAI
- PLLSAI2CFGR PLLSAI2PDIV LL_RCC_PLLSAI2_ConfigDomain_SAI

LL_RCC_PLLSAI2_ConfigDomain_DSI

Function name
__STATIC_INLINE void LL_RCC_PLLSAI2_ConfigDomain_DSI (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLQ)

Function description
Configure PLLSAI2 used for DSI domain clock.

Parameters
- **Source:** This parameter can be one of the following values:
  – LL_RCC_PLLSOURCE_NONE
  – LL_RCC_PLLSOURCE_MSI
  – LL_RCC_PLLSOURCE_HSI
  – LL_RCC_PLLSOURCE_HSE

- **PLLM:** This parameter can be one of the following values:
  - LL_RCC_PLLSAI2M_DIV_1
  - LL_RCC_PLLSAI2M_DIV_2
  - LL_RCC_PLLSAI2M_DIV_3
  - LL_RCC_PLLSAI2M_DIV_4
  - LL_RCC_PLLSAI2M_DIV_5
  - LL_RCC_PLLSAI2M_DIV_6
  - LL_RCC_PLLSAI2M_DIV_7
  - LL_RCC_PLLSAI2M_DIV_8
  - LL_RCC_PLLSAI2M_DIV_9
  - LL_RCC_PLLSAI2M_DIV_10
  - LL_RCC_PLLSAI2M_DIV_11
  - LL_RCC_PLLSAI2M_DIV_12
  - LL_RCC_PLLSAI2M_DIV_13
  - LL_RCC_PLLSAI2M_DIV_14
  - LL_RCC_PLLSAI2M_DIV_15
  - LL_RCC_PLLSAI2M_DIV_16
- **PLLN:** Between 8 and 86
- **PLLQ:** This parameter can be one of the following values:
  - LL_RCC_PLLSAI2Q_DIV_2
  - LL_RCC_PLLSAI2Q_DIV_4
  - LL_RCC_PLLSAI2Q_DIV_6
  - LL_RCC_PLLSAI2Q_DIV_8

Return values
- **None:**

Notes
- PLL Source can be written only when PLL, PLLSAI1 and PLLSAI2 (*) are disabled.
- PLLSAI2M/PLLSAI2N/PLLSAI2Q can be written only when PLLSAI2 is disabled.
- This can be selected for DSI

Reference Manual to LL API cross reference:
- PLLCFGR PLLSRC LL_RCC_PLLSAI2_ConfigDomain_DSI
- PLLSAI2CFGR PLLSAI2M LL_RCC_PLLSAI2_ConfigDomain_DSI
- PLLSAI2CFGR PLLSAI2N LL_RCC_PLLSAI2_ConfigDomain_DSI
- PLLSAI2CFGR PLLSAI2Q LL_RCC_PLLSAI2_ConfigDomain_DSI

LL_RCC_PLLSAI2_ConfigDomain_LTDC

Function name
__STATIC_INLINE void LL_RCC_PLLSAI2_ConfigDomain_LTDC (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR, uint32_t PLLDIVR)

Function description
Configure PLLSAI2 used for LTDC domain clock.

Parameters
- **Source:** This parameter can be one of the following values:
  - LL_RCC_PLLSOURCE_NONE
  - LL_RCC_PLLSOURCE_MSI
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_HSE
- **PLLM:** This parameter can be one of the following values:

- – LL_RCC_PLLSAI2M_DIV_1
- – LL_RCC_PLLSAI2M_DIV_2
- – LL_RCC_PLLSAI2M_DIV_3
- – LL_RCC_PLLSAI2M_DIV_4
- – LL_RCC_PLLSAI2M_DIV_5
- – LL_RCC_PLLSAI2M_DIV_6
- – LL_RCC_PLLSAI2M_DIV_7
- – LL_RCC_PLLSAI2M_DIV_8
- – LL_RCC_PLLSAI2M_DIV_9
- – LL_RCC_PLLSAI2M_DIV_10
- – LL_RCC_PLLSAI2M_DIV_11
- – LL_RCC_PLLSAI2M_DIV_12
- – LL_RCC_PLLSAI2M_DIV_13
- – LL_RCC_PLLSAI2M_DIV_14
- – LL_RCC_PLLSAI2M_DIV_15
- – LL_RCC_PLLSAI2M_DIV_16
- **PLLN:** Between 8 and 86
- **PLLR:** This parameter can be one of the following values:
  - – LL_RCC_PLLSAI2R_DIV_2
  - – LL_RCC_PLLSAI2R_DIV_4
  - – LL_RCC_PLLSAI2R_DIV_6
  - – LL_RCC_PLLSAI2R_DIV_8
- **PLLDIVR:** This parameter can be one of the following values:
  - – LL_RCC_PLLSAI2DIVR_DIV_2
  - – LL_RCC_PLLSAI2DIVR_DIV_4
  - – LL_RCC_PLLSAI2DIVR_DIV_8
  - – LL_RCC_PLLSAI2DIVR_DIV_16

Return values

- **None:**

Notes

- PLL Source can be written only when PLL, PLLSAI1 and PLLSAI2 (*) are disabled.
- PLLSAI2M/PLLSAI2N/PLLSAI2R can be written only when PLLSAI2 is disabled.
- This can be selected for LTDC

Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL_RCC_PLLSAI2_ConfigDomain_LTDC
- PLLSAI2CFGR PLLSAI2M LL_RCC_PLLSAI2_ConfigDomain_LTDC
- PLLSAI2CFGR PLLSAI2N LL_RCC_PLLSAI2_ConfigDomain_LTDC
- PLLSAI2CFGR PLLSAI2R LL_RCC_PLLSAI2_ConfigDomain_LTDC
- CCIPR2 PLLSAI2DIVR LL_RCC_PLLSAI2_ConfigDomain_LTDC

LL_RCC_PLLSAI2_GetN

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI2_GetN (void)**
Function description	Get SAI2PLL multiplication factor for VCO.
Return values	• **Between:** 8 and 86

Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2N LL_RCC_PLLSAI2_GetN

LL_RCC_PLLSAI2_GetP

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI2_GetP (void)**
Function description	Get SAI2PLL division factor for PLLSAI2P.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLLSAI2P_DIV_2 – LL_RCC_PLLSAI2P_DIV_3 – LL_RCC_PLLSAI2P_DIV_4 – LL_RCC_PLLSAI2P_DIV_5 – LL_RCC_PLLSAI2P_DIV_6 – LL_RCC_PLLSAI2P_DIV_7 – LL_RCC_PLLSAI2P_DIV_8 – LL_RCC_PLLSAI2P_DIV_9 – LL_RCC_PLLSAI2P_DIV_10 – LL_RCC_PLLSAI2P_DIV_11 – LL_RCC_PLLSAI2P_DIV_12 – LL_RCC_PLLSAI2P_DIV_13 – LL_RCC_PLLSAI2P_DIV_14 – LL_RCC_PLLSAI2P_DIV_15 – LL_RCC_PLLSAI2P_DIV_16 – LL_RCC_PLLSAI2P_DIV_17 – LL_RCC_PLLSAI2P_DIV_18 – LL_RCC_PLLSAI2P_DIV_19 – LL_RCC_PLLSAI2P_DIV_20 – LL_RCC_PLLSAI2P_DIV_21 – LL_RCC_PLLSAI2P_DIV_22 – LL_RCC_PLLSAI2P_DIV_23 – LL_RCC_PLLSAI2P_DIV_24 – LL_RCC_PLLSAI2P_DIV_25 – LL_RCC_PLLSAI2P_DIV_26 – LL_RCC_PLLSAI2P_DIV_27 – LL_RCC_PLLSAI2P_DIV_28 – LL_RCC_PLLSAI2P_DIV_29 – LL_RCC_PLLSAI2P_DIV_30 – LL_RCC_PLLSAI2P_DIV_31
Notes	• Used for PLLSAI2CLK (SAI1 or SAI2 clock).
Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2PDIV LL_RCC_PLLSAI2_GetP

LL_RCC_PLLSAI2_GetQ

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI2_GetQ (void)**
Function description	Get division factor for PLLSAI2Q.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLLSAI2Q_DIV_2

- – LL_RCC_PLLSAI2Q_DIV_4
- – LL_RCC_PLLSAI2Q_DIV_6
- – LL_RCC_PLLSAI2Q_DIV_8

Notes	• Used for PLLDSICLK (DSI clock)
Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2Q LL_RCC_PLLSAI2_GetQ

LL_RCC_PLLSAI2_GetR

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI2_GetR (void)**
Function description	Get SAI2PLL division factor for PLLSAI2R.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLLSAI2R_DIV_2 – LL_RCC_PLLSAI2R_DIV_4 – LL_RCC_PLLSAI2R_DIV_6 – LL_RCC_PLLSAI2R_DIV_8
Notes	• Used for PLLADC2CLK (ADC clock) or PLLLCDCLK (LTDC clock) depending on devices
Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2R LL_RCC_PLLSAI2_GetR

LL_RCC_PLLSAI2_GetDivider

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI2_GetDivider (void)**
Function description	Get Division factor for the PLLSAI2.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLLSAI2M_DIV_1 – LL_RCC_PLLSAI2M_DIV_2 – LL_RCC_PLLSAI2M_DIV_3 – LL_RCC_PLLSAI2M_DIV_4 – LL_RCC_PLLSAI2M_DIV_5 – LL_RCC_PLLSAI2M_DIV_6 – LL_RCC_PLLSAI2M_DIV_7 – LL_RCC_PLLSAI2M_DIV_8 – LL_RCC_PLLSAI2M_DIV_9 – LL_RCC_PLLSAI2M_DIV_10 – LL_RCC_PLLSAI2M_DIV_11 – LL_RCC_PLLSAI2M_DIV_12 – LL_RCC_PLLSAI2M_DIV_13 – LL_RCC_PLLSAI2M_DIV_14 – LL_RCC_PLLSAI2M_DIV_15 – LL_RCC_PLLSAI2M_DIV_16
Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2M LL_RCC_PLLSAI2_GetDivider

LL_RCC_PLLSAI2_GetDIVR

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLLSAI2_GetDIVR (void)**
Function description	Get PLLSAI2 division factor for PLLSAI2DIVR.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_PLLSAI2DIVR_DIV_2 – LL_RCC_PLLSAI2DIVR_DIV_4 – LL_RCC_PLLSAI2DIVR_DIV_8 – LL_RCC_PLLSAI2DIVR_DIV_16
Notes	• Used for LTDC domain clock
Reference Manual to LL API cross reference:	• CCIPR2 PLLSAI2DIVR LL_RCC_PLLSAI2_GetDIVR

LL_RCC_PLLSAI2_EnableDomain_SAI

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI2_EnableDomain_SAI (void)**
Function description	Enable PLLSAI2 output mapped on SAI domain clock.
Return values	• **None:**
Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2PEN LL_RCC_PLLSAI2_EnableDomain_SAI

LL_RCC_PLLSAI2_DisableDomain_SAI

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI2_DisableDomain_SAI (void)**
Function description	Disable PLLSAI2 output mapped on SAI domain clock.
Return values	• **None:**
Notes	• In order to save power, when of the PLLSAI2 is not used, should be 0
Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2PEN LL_RCC_PLLSAI2_DisableDomain_SAI

LL_RCC_PLLSAI2_EnableDomain_DSI

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI2_EnableDomain_DSI (void)**
Function description	Enable PLLSAI2 output mapped on DSI domain clock.
Return values	• **None:**
Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2QEN LL_RCC_PLLSAI2_EnableDomain_DSI

LL_RCC_PLLSAI2_DisableDomain_DSI

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI2_DisableDomain_DSI (void)**
Function description	Disable PLLSAI2 output mapped on DSI domain clock.
Return values	• **None:**
Notes	• In order to save power, when of the PLLSAI2 is not used, Main PLLSAI2 should be 0
Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2QEN LL_RCC_PLLSAI2_DisableDomain_DSI

LL_RCC_PLLSAI2_EnableDomain_LTDC

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI2_EnableDomain_LTDC (void)**
Function description	Enable PLLSAI2 output mapped on LTDC domain clock.
Return values	• **None:**
Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2REN LL_RCC_PLLSAI2_EnableDomain_LTDC

LL_RCC_PLLSAI2_DisableDomain_LTDC

Function name	**__STATIC_INLINE void LL_RCC_PLLSAI2_DisableDomain_LTDC (void)**
Function description	Disable PLLSAI2 output mapped on LTDC domain clock.
Return values	• **None:**
Notes	• In order to save power, when of the PLLSAI2 is not used, Main PLLSAI2 should be 0
Reference Manual to LL API cross reference:	• PLLSAI2CFGR PLLSAI2REN LL_RCC_PLLSAI2_DisableDomain_LTDC

LL_RCC_ClearFlag_LSIRDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_LSIRDY (void)**
Function description	Clear LSI ready interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR LSIRDYC LL_RCC_ClearFlag_LSIRDY

LL_RCC_ClearFlag_LSERDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_LSERDY (void)**

Function description	Clear LSE ready interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR LSERDYC LL_RCC_ClearFlag_LSERDY

LL_RCC_ClearFlag_MSIRDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_MSIRDY (void)**
Function description	Clear MSI ready interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR MSIRDYC LL_RCC_ClearFlag_MSIRDY

LL_RCC_ClearFlag_HSIRDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_HSIRDY (void)**
Function description	Clear HSI ready interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR HSIRDYC LL_RCC_ClearFlag_HSIRDY

LL_RCC_ClearFlag_HSERDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_HSERDY (void)**
Function description	Clear HSE ready interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR HSERDYC LL_RCC_ClearFlag_HSERDY

LL_RCC_ClearFlag_PLLRDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_PLLRDY (void)**
Function description	Clear PLL ready interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR PLLRDYC LL_RCC_ClearFlag_PLLRDY

LL_RCC_ClearFlag_HSI48RDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_HSI48RDY (void)**
Function description	Clear HSI48 ready interrupt flag.

Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR HSI48RDYC LL_RCC_ClearFlag_HSI48RDY

LL_RCC_ClearFlag_PLLSAI1RDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_PLLSAI1RDY (void)**
Function description	Clear PLLSAI1 ready interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR PLLSAI1RDYC LL_RCC_ClearFlag_PLLSAI1RDY

LL_RCC_ClearFlag_PLLSAI2RDY

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_PLLSAI2RDY (void)**
Function description	Clear PLLSAI1 ready interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR PLLSAI2RDYC LL_RCC_ClearFlag_PLLSAI2RDY

LL_RCC_ClearFlag_HSECSS

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_HSECSS (void)**
Function description	Clear Clock security system interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR CSSC LL_RCC_ClearFlag_HSECSS

LL_RCC_ClearFlag_LSECSS

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_LSECSS (void)**
Function description	Clear LSE Clock security system interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CICR LSECSSC LL_RCC_ClearFlag_LSECSS

LL_RCC_IsActiveFlag_LSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSIRDY (void)**

Function description	Check if LSI ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR LSIRDYF LL_RCC_IsActiveFlag_LSIRDY

LL_RCC_IsActiveFlag_LSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSERDY (void)**
Function description	Check if LSE ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR LSERDYF LL_RCC_IsActiveFlag_LSERDY

LL_RCC_IsActiveFlag_MSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_MSIRDY (void)**
Function description	Check if MSI ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR MSIRDYF LL_RCC_IsActiveFlag_MSIRDY

LL_RCC_IsActiveFlag_HSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIRDY (void)**
Function description	Check if HSI ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR HSIRDYF LL_RCC_IsActiveFlag_HSIRDY

LL_RCC_IsActiveFlag_HSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSERDY (void)**
Function description	Check if HSE ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR HSERDYF LL_RCC_IsActiveFlag_HSERDY

LL_RCC_IsActiveFlag_PLLRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLLRDY (void)**
Function description	Check if PLL ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR PLLRDYF LL_RCC_IsActiveFlag_PLLRDY

LL_RCC_IsActiveFlag_HSI48RDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSI48RDY (void)**
Function description	Check if HSI48 ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR HSI48RDYF LL_RCC_IsActiveFlag_HSI48RDY

LL_RCC_IsActiveFlag_PLLSAI1RDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLLSAI1RDY (void)**
Function description	Check if PLLSAI1 ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR PLLSAI1RDYF LL_RCC_IsActiveFlag_PLLSAI1RDY

LL_RCC_IsActiveFlag_PLLSAI2RDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLLSAI2RDY (void)**
Function description	Check if PLLSAI1 ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR PLLSAI2RDYF LL_RCC_IsActiveFlag_PLLSAI2RDY

LL_RCC_IsActiveFlag_HSECSS

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSECSS (void)**
Function description	Check if Clock security system interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• CIFR CSSF LL_RCC_IsActiveFlag_HSECSS

LL_RCC_IsActiveFlag_LSECSS

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSECSS (void)**
Function description	Check if LSE Clock security system interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIFR LSECSSF LL_RCC_IsActiveFlag_LSECSS

LL_RCC_IsActiveFlag_FWRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_FWRST (void)**
Function description	Check if RCC flag FW reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR FWRSTF LL_RCC_IsActiveFlag_FWRST

LL_RCC_IsActiveFlag_IWDGRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDGRST (void)**
Function description	Check if RCC flag Independent Watchdog reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR IWDGRSTF LL_RCC_IsActiveFlag_IWDGRST

LL_RCC_IsActiveFlag_LPWRRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRRST (void)**
Function description	Check if RCC flag Low Power reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRRST

LL_RCC_IsActiveFlag_OBLRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_OBLRST (void)**

Function description	Check if RCC flag is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR OBLRSTF LL_RCC_IsActiveFlag_OBLRST

LL_RCC_IsActiveFlag_PINRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PINRST (void)**
Function description	Check if RCC flag Pin reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR PINRSTF LL_RCC_IsActiveFlag_PINRST

LL_RCC_IsActiveFlag_SFTRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFTRST (void)**
Function description	Check if RCC flag Software reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SFTRSTF LL_RCC_IsActiveFlag_SFTRST

LL_RCC_IsActiveFlag_WWDGRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WWDGRST (void)**
Function description	Check if RCC flag Window Watchdog reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR WWDGRSTF LL_RCC_IsActiveFlag_WWDGRST

LL_RCC_IsActiveFlag_BORRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_BORRST (void)**
Function description	Check if RCC flag BOR reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR BORRSTF LL_RCC_IsActiveFlag_BORRST

LL_RCC_ClearResetFlags

Function name	**__STATIC_INLINE void LL_RCC_ClearResetFlags (void)**
Function description	Set RMVF bit to clear the reset flags.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR RMVF LL_RCC_ClearResetFlags

LL_RCC_EnableIT_LSIRDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_LSIRDY (void)**
Function description	Enable LSI ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER LSIRDYIE LL_RCC_EnableIT_LSIRDY

LL_RCC_EnableIT_LSERDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_LSERDY (void)**
Function description	Enable LSE ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER LSERDYIE LL_RCC_EnableIT_LSERDY

LL_RCC_EnableIT_MSIRDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_MSIRDY (void)**
Function description	Enable MSI ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER MSIRDYIE LL_RCC_EnableIT_MSIRDY

LL_RCC_EnableIT_HSIRDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_HSIRDY (void)**
Function description	Enable HSI ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER HSIRDYIE LL_RCC_EnableIT_HSIRDY

LL_RCC_EnableIT_HSERDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_HSERDY (void)**
Function description	Enable HSE ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER HSERDYIE LL_RCC_EnableIT_HSERDY

LL_RCC_EnableIT_PLLRDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_PLLRDY (void)**
Function description	Enable PLL ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER PLLRDYIE LL_RCC_EnableIT_PLLRDY

LL_RCC_EnableIT_HSI48RDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_HSI48RDY (void)**
Function description	Enable HSI48 ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER HSI48RDYIE LL_RCC_EnableIT_HSI48RDY

LL_RCC_EnableIT_PLLSAI1RDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_PLLSAI1RDY (void)**
Function description	Enable PLLSAI1 ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER PLLSAI1RDYIE LL_RCC_EnableIT_PLLSAI1RDY

LL_RCC_EnableIT_PLLSAI2RDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_PLLSAI2RDY (void)**
Function description	Enable PLLSAI2 ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER PLLSAI2RDYIE LL_RCC_EnableIT_PLLSAI2RDY

LL_RCC_EnableIT_LSECSS

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_LSECSS (void)**
Function description	Enable LSE clock security system interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER LSECSSIE LL_RCC_EnableIT_LSECSS

LL_RCC_DisableIT_LSIRDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_LSIRDY (void)**
Function description	Disable LSI ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER LSIRDYIE LL_RCC_DisableIT_LSIRDY

LL_RCC_DisableIT_LSERDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_LSERDY (void)**
Function description	Disable LSE ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER LSERDYIE LL_RCC_DisableIT_LSERDY

LL_RCC_DisableIT_MSIRDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_MSIRDY (void)**
Function description	Disable MSI ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER MSIRDYIE LL_RCC_DisableIT_MSIRDY

LL_RCC_DisableIT_HSIRDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_HSIRDY (void)**
Function description	Disable HSI ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER HSIRDYIE LL_RCC_DisableIT_HSIRDY

LL_RCC_DisableIT_HSERDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_HSERDY (void)**
Function description	Disable HSE ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER HSERDYIE LL_RCC_DisableIT_HSERDY

LL_RCC_DisableIT_PLLRDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_PLLRDY (void)**
Function description	Disable PLL ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER PLLRDYIE LL_RCC_DisableIT_PLLRDY

LL_RCC_DisableIT_HSI48RDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_HSI48RDY (void)**
Function description	Disable HSI48 ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER HSI48RDYIE LL_RCC_DisableIT_HSI48RDY

LL_RCC_DisableIT_PLLSAI1RDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_PLLSAI1RDY (void)**
Function description	Disable PLLSAI1 ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER PLLSAI1RDYIE LL_RCC_DisableIT_PLLSAI1RDY

LL_RCC_DisableIT_PLLSAI2RDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_PLLSAI2RDY (void)**
Function description	Disable PLLSAI2 ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER PLLSAI2RDYIE LL_RCC_DisableIT_PLLSAI2RDY

LL_RCC_DisableIT_LSECSS

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_LSECSS (void)**
Function description	Disable LSE clock security system interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIER LSECSSIE LL_RCC_DisableIT_LSECSS

LL_RCC_IsEnabledIT_LSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSIRDY (void)**
Function description	Checks if LSI ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER LSIRDYIE LL_RCC_IsEnabledIT_LSIRDY

LL_RCC_IsEnabledIT_LSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSERDY (void)**
Function description	Checks if LSE ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER LSERDYIE LL_RCC_IsEnabledIT_LSERDY

LL_RCC_IsEnabledIT_MSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_MSIRDY (void)**
Function description	Checks if MSI ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER MSIRDYIE LL_RCC_IsEnabledIT_MSIRDY

LL_RCC_IsEnabledIT_HSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSIRDY (void)**
Function description	Checks if HSI ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CIER HSIRDYIE LL_RCC_IsEnabledIT_HSIRDY

reference:

LL_RCC_IsEnabledIT_HSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSERDY (void)**
Function description	Checks if HSE ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER HSERDYIE LL_RCC_IsEnabledIT_HSERDY

LL_RCC_IsEnabledIT_PLLRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLLRDY (void)**
Function description	Checks if PLL ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER PLLRDYIE LL_RCC_IsEnabledIT_PLLRDY

LL_RCC_IsEnabledIT_HSI48RDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSI48RDY (void)**
Function description	Checks if HSI48 ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER HSI48RDYIE LL_RCC_IsEnabledIT_HSI48RDY

LL_RCC_IsEnabledIT_PLLSAI1RDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLLSAI1RDY (void)**
Function description	Checks if PLLSAI1 ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER PLLSAI1RDYIE LL_RCC_IsEnabledIT_PLLSAI1RDY

LL_RCC_IsEnabledIT_PLLSAI2RDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLLSAI2RDY (void)**
Function description	Checks if PLLSAI2 ready interrupt source is enabled or disabled.

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER PLLSAI2RDYIE LL_RCC_IsEnabledIT_PLLSAI2RDY

LL_RCC_IsEnabledIT_LSECSS

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSECSS (void)**
Function description	Checks if LSECSS interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIER LSECSSIE LL_RCC_IsEnabledIT_LSECSS

LL_RCC_DeInit

Function name	**ErrorStatus LL_RCC_DeInit (void)**
Function description	Reset the RCC clock configuration to the default reset state.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RCC registers are de-initialized – ERROR: not applicable
Notes	• The default reset state of the clock configuration is given below: MSI ON and used as system clock sourceHSE, HSI, PLL and PLLSAIxSource OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO OFFAll interrupts disabled • This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

LL_RCC_GetSystemClocksFreq

Function name	**void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)**
Function description	Return the frequencies of different on chip clocks; System, AHB, APB1 and APB2 buses clocks.
Parameters	• **RCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies
Return values	• **None:**
Notes	• Each time SYSCLK, HCLK, PCLK1 and/or PCLK2 clock changes, this function must be called to update structure fields. Otherwise, any configuration based on this function will be incorrect.

LL_RCC_GetUSARTClockFreq

Function name	**uint32_t LL_RCC_GetUSARTClockFreq (uint32_t USARTxSource)**

Function description	Return USARTx clock frequency.
Parameters	• **USARTxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_USART1_CLKSOURCE – LL_RCC_USART2_CLKSOURCE – LL_RCC_USART3_CLKSOURCE (*)
Return values	• **USART:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetUARTClockFreq

Function name	**uint32_t LL_RCC_GetUARTClockFreq (uint32_t UARTxSource)**
Function description	Return UARTx clock frequency.
Parameters	• **UARTxSource:** This parameter can be one of the following values: – LL_RCC_UART4_CLKSOURCE – LL_RCC_UART5_CLKSOURCE
Return values	• **UART:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetI2CClockFreq

Function name	**uint32_t LL_RCC_GetI2CClockFreq (uint32_t I2CxSource)**
Function description	Return I2Cx clock frequency.
Parameters	• **I2CxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_I2C1_CLKSOURCE – LL_RCC_I2C2_CLKSOURCE (*) – LL_RCC_I2C3_CLKSOURCE – LL_RCC_I2C4_CLKSOURCE (*)
Return values	• **I2C:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that HSI oscillator is not ready

LL_RCC_GetLPUARTClockFreq

Function name	**uint32_t LL_RCC_GetLPUARTClockFreq (uint32_t LPUARTxSource)**
Function description	Return LPUARTx clock frequency.
Parameters	• **LPUARTxSource:** This parameter can be one of the following values: – LL_RCC_LPUART1_CLKSOURCE
Return values	• **LPUART:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that

oscillator (HSI or LSE) is not ready

LL_RCC_GetLPTIMClockFreq

Function name	**uint32_t LL_RCC_GetLPTIMClockFreq (uint32_t LPTIMxSource)**
Function description	Return LPTIMx clock frequency.
Parameters	• **LPTIMxSource:** This parameter can be one of the following values: – LL_RCC_LPTIM1_CLKSOURCE – LL_RCC_LPTIM2_CLKSOURCE
Return values	• **LPTIM:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI, LSI or LSE) is not ready

LL_RCC_GetSAIClockFreq

Function name	**uint32_t LL_RCC_GetSAIClockFreq (uint32_t SAIxSource)**
Function description	Return SAIx clock frequency.
Parameters	• **SAIxSource:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_RCC_SAI1_CLKSOURCE – LL_RCC_SAI2_CLKSOURCE (*)
Return values	• **SAI:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that PLL is not ready – LL_RCC_PERIPH_FREQUENCY_NA indicates that external clock is used

LL_RCC_GetSDMMCClockFreq

Function name	**uint32_t LL_RCC_GetSDMMCClockFreq (uint32_t SDMMCxSource)**
Function description	Return SDMMCx clock frequency.
Parameters	• **SDMMCxSource:** This parameter can be one of the following values: – LL_RCC_SDMMC1_CLKSOURCE
Return values	• **SDMMC:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (MSI) or PLL is not ready – LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

LL_RCC_GetRNGClockFreq

Function name	**uint32_t LL_RCC_GetRNGClockFreq (uint32_t RNGxSource)**
Function description	Return RNGx clock frequency.
Parameters	• **RNGxSource:** This parameter can be one of the following

values:
- LL_RCC_RNG_CLKSOURCE

Return values	• **RNG:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (MSI) or PLL is not ready – LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

LL_RCC_GetUSBClockFreq

Function name	**uint32_t LL_RCC_GetUSBClockFreq (uint32_t USBxSource)**
Function description	Return USBx clock frequency.
Parameters	• **USBxSource:** This parameter can be one of the following values: – LL_RCC_USB_CLKSOURCE
Return values	• **USB:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (MSI) or PLL is not ready – LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

LL_RCC_GetADCClockFreq

Function name	**uint32_t LL_RCC_GetADCClockFreq (uint32_t ADCxSource)**
Function description	Return ADCx clock frequency.
Parameters	• **ADCxSource:** This parameter can be one of the following values: – LL_RCC_ADC_CLKSOURCE
Return values	• **ADC:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (MSI) or PLL is not ready – LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

LL_RCC_GetDFSDMClockFreq

Function name	**uint32_t LL_RCC_GetDFSDMClockFreq (uint32_t DFSDMxSource)**
Function description	Return DFSDMx clock frequency.
Parameters	• **DFSDMxSource:** This parameter can be one of the following values: – LL_RCC_DFSDM1_CLKSOURCE
Return values	• **DFSDM:** clock frequency (in Hz)

LL_RCC_GetDFSDMAudioClockFreq

Function name	**uint32_t LL_RCC_GetDFSDMAudioClockFreq (uint32_t DFSDMxSource)**

Function description	Return DFSDMx Audio clock frequency.
Parameters	• **DFSDMxSource:** This parameter can be one of the following values: – LL_RCC_DFSDM1_AUDIO_CLKSOURCE
Return values	• **DFSDM:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetLTDCClockFreq

Function name	**uint32_t LL_RCC_GetLTDCClockFreq (uint32_t LTDCxSource)**
Function description	Return LTDC clock frequency.
Parameters	• **LTDCxSource:** This parameter can be one of the following values: – LL_RCC_LTDC_CLKSOURCE
Return values	• **LTDC:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator PLLSAI is not ready

LL_RCC_GetDSIClockFreq

Function name	**uint32_t LL_RCC_GetDSIClockFreq (uint32_t DSIxSource)**
Function description	Return DSI clock frequency.
Parameters	• **DSIxSource:** This parameter can be one of the following values: – LL_RCC_DSI_CLKSOURCE
Return values	• **DSI:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready – LL_RCC_PERIPH_FREQUENCY_NA indicates that external clock is used

LL_RCC_GetOCTOSPIClockFreq

Function name	**uint32_t LL_RCC_GetOCTOSPIClockFreq (uint32_t OCTOSPIxSource)**
Function description	Return OCTOSPI clock frequency.
Parameters	• **OCTOSPIxSource:** This parameter can be one of the following values: – LL_RCC_OCTOSPI_CLKSOURCE
Return values	• **OCTOSPI:** clock frequency (in Hz) – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator PLLSAI is not ready

91.3 RCC Firmware driver defines

91.3.1 RCC

Peripheral ADC get clock source

LL_RCC_ADC_CLKSOURCE ADC Clock source selection

Peripheral ADC clock source selection

LL_RCC_ADC_CLKSOURCE_NONE No clock used as ADC clock source

LL_RCC_ADC_CLKSOURCE_PLLSAI1 PLLSAI1 clock used as ADC clock source

LL_RCC_ADC_CLKSOURCE_SYSCLK SYSCLK clock used as ADC clock source

APB low-speed prescaler (APB1)

LL_RCC_APB1_DIV_1 HCLK not divided

LL_RCC_APB1_DIV_2 HCLK divided by 2

LL_RCC_APB1_DIV_4 HCLK divided by 4

LL_RCC_APB1_DIV_8 HCLK divided by 8

LL_RCC_APB1_DIV_16 HCLK divided by 16

APB high-speed prescaler (APB2)

LL_RCC_APB2_DIV_1 HCLK not divided

LL_RCC_APB2_DIV_2 HCLK divided by 2

LL_RCC_APB2_DIV_4 HCLK divided by 4

LL_RCC_APB2_DIV_8 HCLK divided by 8

LL_RCC_APB2_DIV_16 HCLK divided by 16

Clear Flags Defines

LL_RCC_CICR_LSIRDYC LSI Ready Interrupt Clear

LL_RCC_CICR_LSERDYC LSE Ready Interrupt Clear

LL_RCC_CICR_MSIRDYC MSI Ready Interrupt Clear

LL_RCC_CICR_HSIRDYC HSI Ready Interrupt Clear

LL_RCC_CICR_HSERDYC HSE Ready Interrupt Clear

LL_RCC_CICR_PLLRDYC PLL Ready Interrupt Clear

LL_RCC_CICR_HSI48RDYC HSI48 Ready Interrupt Clear

LL_RCC_CICR_PLLSAI1RDYC PLLSAI1 Ready Interrupt Clear

LL_RCC_CICR_PLLSAI2RDYC PLLSAI2 Ready Interrupt Clear

LL_RCC_CICR_LSECSSC LSE Clock Security System Interrupt Clear

LL_RCC_CICR_CSSC Clock Security System Interrupt Clear

Peripheral DFSDM1 get clock source

LL_RCC_DFSDM1_CLKSOURCE DFSDM1 Clock source selection

Peripheral DFSDM1 Audio get clock source

LL_RCC_DFSDM1_AUDIO_CLKSOURCE

Peripheral DFSDM1 Audio clock source selection

LL_RCC_DFSDM1_AUDIO_CLKSOURCE_SAI1	SAI1 clock used as DFSDM1 Audio clock
LL_RCC_DFSDM1_AUDIO_CLKSOURCE_HSI	HSI clock used as DFSDM1 Audio clock
LL_RCC_DFSDM1_AUDIO_CLKSOURCE_MSI	MSI clock used as DFSDM1 Audio clock

Peripheral DFSDM1 clock source selection

LL_RCC_DFSDM1_CLKSOURCE_PCLK2	PCLK2 used as DFSDM1 clock source
LL_RCC_DFSDM1_CLKSOURCE_SYSCLK	SYSCLK used as DFSDM1 clock source

Peripheral DSI get clock source

LL_RCC_DSI_CLKSOURCE	DSI Clock source selection

Peripheral DSI clock source selection

LL_RCC_DSI_CLKSOURCE_PHY	DSI-PHY clock used as DSI byte lane clock source
LL_RCC_DSI_CLKSOURCE_PLL	PLL clock used as DSI byte lane clock source

Get Flags Defines

LL_RCC_CIFR_LSIRDYF	LSI Ready Interrupt flag
LL_RCC_CIFR_LSERDYF	LSE Ready Interrupt flag
LL_RCC_CIFR_MSIRDYF	MSI Ready Interrupt flag
LL_RCC_CIFR_HSIRDYF	HSI Ready Interrupt flag
LL_RCC_CIFR_HSERDYF	HSE Ready Interrupt flag
LL_RCC_CIFR_PLLRDYF	PLL Ready Interrupt flag
LL_RCC_CIFR_HSI48RDYF	HSI48 Ready Interrupt flag
LL_RCC_CIFR_PLLSAI1RDYF	PLLSAI1 Ready Interrupt flag
LL_RCC_CIFR_PLLSAI2RDYF	PLLSAI2 Ready Interrupt flag
LL_RCC_CIFR_LSECSSF	LSE Clock Security System Interrupt flag
LL_RCC_CIFR_CSSF	Clock Security System Interrupt flag
LL_RCC_CSR_FWRSTF	Firewall reset flag
LL_RCC_CSR_LPWRRSTF	Low-Power reset flag
LL_RCC_CSR_OBLRSTF	OBL reset flag
LL_RCC_CSR_PINRSTF	PIN reset flag
LL_RCC_CSR_SFTRSTF	Software Reset flag
LL_RCC_CSR_IWDGRSTF	Independent Watchdog reset flag
LL_RCC_CSR_WWDGRSTF	Window watchdog reset flag
LL_RCC_CSR_BORRSTF	BOR reset flag

Peripheral I2C get clock source

LL_RCC_I2C1_CLKSOURCE	I2C1 Clock source selection

LL_RCC_I2C2_CLKSOURCE	I2C2 Clock source selection
LL_RCC_I2C3_CLKSOURCE	I2C3 Clock source selection
LL_RCC_I2C4_CLKSOURCE	I2C4 Clock source selection

Peripheral I2C clock source selection

LL_RCC_I2C1_CLKSOURCE_PCLK1	PCLK1 clock used as I2C1 clock source
LL_RCC_I2C1_CLKSOURCE_SYSCLK	SYSCLK clock used as I2C1 clock source
LL_RCC_I2C1_CLKSOURCE_HSI	HSI clock used as I2C1 clock source
LL_RCC_I2C2_CLKSOURCE_PCLK1	PCLK1 clock used as I2C2 clock source
LL_RCC_I2C2_CLKSOURCE_SYSCLK	SYSCLK clock used as I2C2 clock source
LL_RCC_I2C2_CLKSOURCE_HSI	HSI clock used as I2C2 clock source
LL_RCC_I2C3_CLKSOURCE_PCLK1	PCLK1 clock used as I2C3 clock source
LL_RCC_I2C3_CLKSOURCE_SYSCLK	SYSCLK clock used as I2C3 clock source
LL_RCC_I2C3_CLKSOURCE_HSI	HSI clock used as I2C3 clock source
LL_RCC_I2C4_CLKSOURCE_PCLK1	PCLK1 clock used as I2C4 clock source
LL_RCC_I2C4_CLKSOURCE_SYSCLK	SYSCLK clock used as I2C4 clock source
LL_RCC_I2C4_CLKSOURCE_HSI	HSI clock used as I2C4 clock source

IT Defines

LL_RCC_CIER_LSIRDYIE	LSI Ready Interrupt Enable
LL_RCC_CIER_LSERDYIE	LSE Ready Interrupt Enable
LL_RCC_CIER_MSIRDYIE	MSI Ready Interrupt Enable
LL_RCC_CIER_HSIRDYIE	HSI Ready Interrupt Enable
LL_RCC_CIER_HSERDYIE	HSE Ready Interrupt Enable
LL_RCC_CIER_PLLRDYIE	PLL Ready Interrupt Enable
LL_RCC_CIER_HSI48RDYIE	HSI48 Ready Interrupt Enable
LL_RCC_CIER_PLLSAI1RDYIE	PLLSAI1 Ready Interrupt Enable
LL_RCC_CIER_PLLSAI2RDYIE	PLLSAI2 Ready Interrupt Enable
LL_RCC_CIER_LSECSSIE	LSE CSS Interrupt Enable

Peripheral LPTIM get clock source

LL_RCC_LPTIM1_CLKSOURCE	LPTIM1 Clock source selection
LL_RCC_LPTIM2_CLKSOURCE	LPTIM2 Clock source selection

Peripheral LPTIM clock source selection

LL_RCC_LPTIM1_CLKSOURCE_PCLK1	PCLK1 clock used as LPTIM1 clock source
LL_RCC_LPTIM1_CLKSOURCE_LSI	LSI clock used as LPTIM1 clock source
LL_RCC_LPTIM1_CLKSOURCE_HSI	HSI clock used as LPTIM1 clock source
LL_RCC_LPTIM1_CLKSOURCE_LSE	LSE clock used as LPTIM1 clock source
LL_RCC_LPTIM2_CLKSOURCE_PCLK1	PCLK1 clock used as LPTIM2 clock source

LL_RCC_LPTIM2_CLKSOURCE_LSI	LSI clock used as LPTIM2 clock source
LL_RCC_LPTIM2_CLKSOURCE_HSI	HSI clock used as LPTIM2 clock source
LL_RCC_LPTIM2_CLKSOURCE_LSE	LSE clock used as LPTIM2 clock source

Peripheral LPUART get clock source

LL_RCC_LPUART1_CLKSOURCE	LPUART1 Clock source selection

Peripheral LPUART clock source selection

LL_RCC_LPUART1_CLKSOURCE_PCLK1	PCLK1 clock used as LPUART1 clock source
LL_RCC_LPUART1_CLKSOURCE_SYSCLK	SYSCLK clock used as LPUART1 clock source
LL_RCC_LPUART1_CLKSOURCE_HSI	HSI clock used as LPUART1 clock source
LL_RCC_LPUART1_CLKSOURCE_LSE	LSE clock used as LPUART1 clock source

LSCO Selection

LL_RCC_LSCO_CLKSOURCE_LSI	LSI selection for low speed clock
LL_RCC_LSCO_CLKSOURCE_LSE	LSE selection for low speed clock

LSE oscillator drive capability

LL_RCC_LSEDRIVE_LOW	Xtal mode lower driving capability
LL_RCC_LSEDRIVE_MEDIUMLOW	Xtal mode medium low driving capability
LL_RCC_LSEDRIVE_MEDIUMHIGH	Xtal mode medium high driving capability
LL_RCC_LSEDRIVE_HIGH	Xtal mode higher driving capability

Peripheral LTDC get clock source

LL_RCC_LTDC_CLKSOURCE	LTDC Clock source selection

Peripheral LTDC clock source selection

LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV2	PLLSAI2DIVR divided by 2 used as LTDC clock source
LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV4	PLLSAI2DIVR divided by 4 used as LTDC clock source
LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV8	PLLSAI2DIVR divided by 8 used as LTDC clock source
LL_RCC_LTDC_CLKSOURCE_PLLSAI2R_DIV16	PLLSAI2DIVR divided by 16 used as LTDC clock source

MCO1 SOURCE selection

LL_RCC_MCO1SOURCE_NOCLOCK	MCO output disabled, no clock on MCO
LL_RCC_MCO1SOURCE_SYSCLK	SYSCLK selection as MCO1 source
LL_RCC_MCO1SOURCE_MSI	MSI selection as MCO1 source
LL_RCC_MCO1SOURCE_HSI	HSI16 selection as MCO1 source
LL_RCC_MCO1SOURCE_HSE	HSE selection as MCO1 source
LL_RCC_MCO1SOURCE_PLLCLK	Main PLL selection as MCO1 source

LL_RCC_MCO1SOURCE_LSI	LSI selection as MCO1 source
LL_RCC_MCO1SOURCE_LSE	LSE selection as MCO1 source
LL_RCC_MCO1SOURCE_HSI48	HSI48 selection as MCO1 source

MCO1 prescaler

LL_RCC_MCO1_DIV_1	MCO not divided
LL_RCC_MCO1_DIV_2	MCO divided by 2
LL_RCC_MCO1_DIV_4	MCO divided by 4
LL_RCC_MCO1_DIV_8	MCO divided by 8
LL_RCC_MCO1_DIV_16	MCO divided by 16

MSI clock ranges

LL_RCC_MSIRANGE_0	MSI = 100 KHz
LL_RCC_MSIRANGE_1	MSI = 200 KHz
LL_RCC_MSIRANGE_2	MSI = 400 KHz
LL_RCC_MSIRANGE_3	MSI = 800 KHz
LL_RCC_MSIRANGE_4	MSI = 1 MHz
LL_RCC_MSIRANGE_5	MSI = 2 MHz
LL_RCC_MSIRANGE_6	MSI = 4 MHz
LL_RCC_MSIRANGE_7	MSI = 8 MHz
LL_RCC_MSIRANGE_8	MSI = 16 MHz
LL_RCC_MSIRANGE_9	MSI = 24 MHz
LL_RCC_MSIRANGE_10	MSI = 32 MHz
LL_RCC_MSIRANGE_11	MSI = 48 MHz

MSI clock range selection

LL_RCC_MSIRANGESEL_STANDBY	MSI Range is provided by MSISRANGE
LL_RCC_MSIRANGESEL_RUN	MSI Range is provided by MSIRANGE

MSI range after Standby mode

LL_RCC_MSISRANGE_4	MSI = 1 MHz
LL_RCC_MSISRANGE_5	MSI = 2 MHz
LL_RCC_MSISRANGE_6	MSI = 4 MHz
LL_RCC_MSISRANGE_7	MSI = 8 MHz

Peripheral OCTOSPI get clock source

LL_RCC_OCTOSPI_CLKSOURCE_SYSCLK	SYSCLK used as OctoSPI clock source
LL_RCC_OCTOSPI_CLKSOURCE_MSI	MSI used as OctoSPI clock source
LL_RCC_OCTOSPI_CLKSOURCE_PLL	PLL used as OctoSPI clock source
LL_RCC_OCTOSPI_CLKSOURCE	OctoSPI Clock source selection

Oscillator Values adaptation

HSE_VALUE	Value of the HSE oscillator in Hz
HSI_VALUE	Value of the HSI oscillator in Hz
LSE_VALUE	Value of the LSE oscillator in Hz
LSI_VALUE	Value of the LSI oscillator in Hz
HSI48_VALUE	Value of the HSI48 oscillator in Hz

Peripheral clock frequency

LL_RCC_PERIPH_FREQUENCY_NO	No clock enabled for the peripheral
LL_RCC_PERIPH_FREQUENCY_NA	Frequency cannot be provided as external clock

PLL division factor

LL_RCC_PLLM_DIV_1	Main PLL division factor for PLLM input by 1
LL_RCC_PLLM_DIV_2	Main PLL division factor for PLLM input by 2
LL_RCC_PLLM_DIV_3	Main PLL division factor for PLLM input by 3
LL_RCC_PLLM_DIV_4	Main PLL division factor for PLLM input by 4
LL_RCC_PLLM_DIV_5	Main PLL division factor for PLLM input by 5
LL_RCC_PLLM_DIV_6	Main PLL division factor for PLLM input by 6
LL_RCC_PLLM_DIV_7	Main PLL division factor for PLLM input by 7
LL_RCC_PLLM_DIV_8	Main PLL division factor for PLLM input by 8
LL_RCC_PLLM_DIV_9	Main PLL division factor for PLLM input by 9
LL_RCC_PLLM_DIV_10	Main PLL division factor for PLLM input by 10
LL_RCC_PLLM_DIV_11	Main PLL division factor for PLLM input by 11
LL_RCC_PLLM_DIV_12	Main PLL division factor for PLLM input by 12
LL_RCC_PLLM_DIV_13	Main PLL division factor for PLLM input by 13
LL_RCC_PLLM_DIV_14	Main PLL division factor for PLLM input by 14
LL_RCC_PLLM_DIV_15	Main PLL division factor for PLLM input by 15
LL_RCC_PLLM_DIV_16	Main PLL division factor for PLLM input by 16

PLL division factor (PLLP)

LL_RCC_PLLP_DIV_2	Main PLL division factor for PLLP output by 2
LL_RCC_PLLP_DIV_3	Main PLL division factor for PLLP output by 3
LL_RCC_PLLP_DIV_4	Main PLL division factor for PLLP output by 4
LL_RCC_PLLP_DIV_5	Main PLL division factor for PLLP output by 5
LL_RCC_PLLP_DIV_6	Main PLL division factor for PLLP output by 6
LL_RCC_PLLP_DIV_7	Main PLL division factor for PLLP output by 7
LL_RCC_PLLP_DIV_8	Main PLL division factor for PLLP output by 8
LL_RCC_PLLP_DIV_9	Main PLL division factor for PLLP output by 9
LL_RCC_PLLP_DIV_10	Main PLL division factor for PLLP output by 10
LL_RCC_PLLP_DIV_11	Main PLL division factor for PLLP output by 11

LL_RCC_PLLP_DIV_12	Main PLL division factor for PLLP output by 12
LL_RCC_PLLP_DIV_13	Main PLL division factor for PLLP output by 13
LL_RCC_PLLP_DIV_14	Main PLL division factor for PLLP output by 14
LL_RCC_PLLP_DIV_15	Main PLL division factor for PLLP output by 15
LL_RCC_PLLP_DIV_16	Main PLL division factor for PLLP output by 16
LL_RCC_PLLP_DIV_17	Main PLL division factor for PLLP output by 17
LL_RCC_PLLP_DIV_18	Main PLL division factor for PLLP output by 18
LL_RCC_PLLP_DIV_19	Main PLL division factor for PLLP output by 19
LL_RCC_PLLP_DIV_20	Main PLL division factor for PLLP output by 20
LL_RCC_PLLP_DIV_21	Main PLL division factor for PLLP output by 21
LL_RCC_PLLP_DIV_22	Main PLL division factor for PLLP output by 22
LL_RCC_PLLP_DIV_23	Main PLL division factor for PLLP output by 23
LL_RCC_PLLP_DIV_24	Main PLL division factor for PLLP output by 24
LL_RCC_PLLP_DIV_25	Main PLL division factor for PLLP output by 25
LL_RCC_PLLP_DIV_26	Main PLL division factor for PLLP output by 26
LL_RCC_PLLP_DIV_27	Main PLL division factor for PLLP output by 27
LL_RCC_PLLP_DIV_28	Main PLL division factor for PLLP output by 28
LL_RCC_PLLP_DIV_29	Main PLL division factor for PLLP output by 29
LL_RCC_PLLP_DIV_30	Main PLL division factor for PLLP output by 30
LL_RCC_PLLP_DIV_31	Main PLL division factor for PLLP output by 31

PLL division factor (PLLQ)

LL_RCC_PLLQ_DIV_2	Main PLL division factor for PLLQ output by 2
LL_RCC_PLLQ_DIV_4	Main PLL division factor for PLLQ output by 4
LL_RCC_PLLQ_DIV_6	Main PLL division factor for PLLQ output by 6
LL_RCC_PLLQ_DIV_8	Main PLL division factor for PLLQ output by 8

PLL division factor (PLLR)

LL_RCC_PLLR_DIV_2	Main PLL division factor for PLLCLK (system clock) by 2
LL_RCC_PLLR_DIV_4	Main PLL division factor for PLLCLK (system clock) by 4
LL_RCC_PLLR_DIV_6	Main PLL division factor for PLLCLK (system clock) by 6
LL_RCC_PLLR_DIV_8	Main PLL division factor for PLLCLK (system clock) by 8

PLLSAI1 division factor (PLLSAI1M)

LL_RCC_PLLSAI1M_DIV_1	PLLSAI1 division factor for PLLSAI1M input by 1
LL_RCC_PLLSAI1M_DIV_2	PLLSAI1 division factor for PLLSAI1M input by 2
LL_RCC_PLLSAI1M_DIV_3	PLLSAI1 division factor for PLLSAI1M input by 3
LL_RCC_PLLSAI1M_DIV_4	PLLSAI1 division factor for PLLSAI1M input by 4
LL_RCC_PLLSAI1M_DIV_5	PLLSAI1 division factor for PLLSAI1M input by 5

LL_RCC_PLLSAI1M_DIV_6	PLLSAI1 division factor for PLLSAI1M input by 6
LL_RCC_PLLSAI1M_DIV_7	PLLSAI1 division factor for PLLSAI1M input by 7
LL_RCC_PLLSAI1M_DIV_8	PLLSAI1 division factor for PLLSAI1M input by 8
LL_RCC_PLLSAI1M_DIV_9	PLLSAI1 division factor for PLLSAI1M input by 9
LL_RCC_PLLSAI1M_DIV_10	PLLSAI1 division factor for PLLSAI1M input by 10
LL_RCC_PLLSAI1M_DIV_11	PLLSAI1 division factor for PLLSAI1M input by 11
LL_RCC_PLLSAI1M_DIV_12	PLLSAI1 division factor for PLLSAI1M input by 12
LL_RCC_PLLSAI1M_DIV_13	PLLSAI1 division factor for PLLSAI1M input by 13
LL_RCC_PLLSAI1M_DIV_14	PLLSAI1 division factor for PLLSAI1M input by 14
LL_RCC_PLLSAI1M_DIV_15	PLLSAI1 division factor for PLLSAI1M input by 15
LL_RCC_PLLSAI1M_DIV_16	PLLSAI1 division factor for PLLSAI1M input by 16

PLLSAI1 division factor (PLLSAI1P)

LL_RCC_PLLSAI1P_DIV_2	PLLSAI1 division factor for PLLSAI1P output by 2
LL_RCC_PLLSAI1P_DIV_3	PLLSAI1 division factor for PLLSAI1P output by 3
LL_RCC_PLLSAI1P_DIV_4	PLLSAI1 division factor for PLLSAI1P output by 4
LL_RCC_PLLSAI1P_DIV_5	PLLSAI1 division factor for PLLSAI1P output by 5
LL_RCC_PLLSAI1P_DIV_6	PLLSAI1 division factor for PLLSAI1P output by 6
LL_RCC_PLLSAI1P_DIV_7	PLLSAI1 division factor for PLLSAI1P output by 7
LL_RCC_PLLSAI1P_DIV_8	PLLSAI1 division factor for PLLSAI1P output by 8
LL_RCC_PLLSAI1P_DIV_9	PLLSAI1 division factor for PLLSAI1P output by 9
LL_RCC_PLLSAI1P_DIV_10	PLLSAI1 division factor for PLLSAI1P output by 10
LL_RCC_PLLSAI1P_DIV_11	PLLSAI1 division factor for PLLSAI1P output by 1
LL_RCC_PLLSAI1P_DIV_12	PLLSAI1 division factor for PLLSAI1P output by 12
LL_RCC_PLLSAI1P_DIV_13	PLLSAI1 division factor for PLLSAI1P output by 13
LL_RCC_PLLSAI1P_DIV_14	PLLSAI1 division factor for PLLSAI1P output by 14
LL_RCC_PLLSAI1P_DIV_15	PLLSAI1 division factor for PLLSAI1P output by 15
LL_RCC_PLLSAI1P_DIV_16	PLLSAI1 division factor for PLLSAI1P output by 16
LL_RCC_PLLSAI1P_DIV_17	PLLSAI1 division factor for PLLSAI1P output by 17
LL_RCC_PLLSAI1P_DIV_18	PLLSAI1 division factor for PLLSAI1P output by 18
LL_RCC_PLLSAI1P_DIV_19	PLLSAI1 division factor for PLLSAI1P output by 19
LL_RCC_PLLSAI1P_DIV_20	PLLSAI1 division factor for PLLSAI1P output by 20
LL_RCC_PLLSAI1P_DIV_21	PLLSAI1 division fctor for PLLSAI1P output by 21
LL_RCC_PLLSAI1P_DIV_22	PLLSAI1 division factor for PLLSAI1P output by 22
LL_RCC_PLLSAI1P_DIV_23	PLLSAI1 division factor for PLLSAI1P output by 23
LL_RCC_PLLSAI1P_DIV_24	PLLSAI1 division factor for PLLSAI1P output by 24
LL_RCC_PLLSAI1P_DIV_25	PLLSAI1 division factor for PLLSAI1P output by 25

LL_RCC_PLLSAI1P_DIV_26	PLLSAI1 division factor for PLLSAI1P output by 26
LL_RCC_PLLSAI1P_DIV_27	PLLSAI1 division factor for PLLSAI1P output by 27
LL_RCC_PLLSAI1P_DIV_28	PLLSAI1 division factor for PLLSAI1P output by 28
LL_RCC_PLLSAI1P_DIV_29	PLLSAI1 division factor for PLLSAI1P output by 29
LL_RCC_PLLSAI1P_DIV_30	PLLSAI1 division factor for PLLSAI1P output by 30
LL_RCC_PLLSAI1P_DIV_31	PLLSAI1 division factor for PLLSAI1P output by 31

PLLSAI1 division factor (PLLSAI1Q)

LL_RCC_PLLSAI1Q_DIV_2	PLLSAI1 division factor for PLLSAI1Q output by 2
LL_RCC_PLLSAI1Q_DIV_4	PLLSAI1 division factor for PLLSAI1Q output by 4
LL_RCC_PLLSAI1Q_DIV_6	PLLSAI1 division factor for PLLSAI1Q output by 6
LL_RCC_PLLSAI1Q_DIV_8	PLLSAI1 division factor for PLLSAI1Q output by 8

PLLSAI1 division factor (PLLSAI1R)

LL_RCC_PLLSAI1R_DIV_2	PLLSAI1 division factor for PLLSAI1R output by 2
LL_RCC_PLLSAI1R_DIV_4	PLLSAI1 division factor for PLLSAI1R output by 4
LL_RCC_PLLSAI1R_DIV_6	PLLSAI1 division factor for PLLSAI1R output by 6
LL_RCC_PLLSAI1R_DIV_8	PLLSAI1 division factor for PLLSAI1R output by 8

PLLSAI2DIVR division factor (PLLSAI2DIVR)

LL_RCC_PLLSAI2DIVR_DIV_2	PLLSAI2 division factor for PLLSAI2DIVR output by 2
LL_RCC_PLLSAI2DIVR_DIV_4	PLLSAI2 division factor for PLLSAI2DIVR output by 4
LL_RCC_PLLSAI2DIVR_DIV_8	PLLSAI2 division factor for PLLSAI2DIVR output by 8
LL_RCC_PLLSAI2DIVR_DIV_16	PLLSAI2 division factor for PLLSAI2DIVR output by 16

PLLSAI1 division factor (PLLSAI2M)

LL_RCC_PLLSAI2M_DIV_1	PLLSAI2 division factor for PLLSAI2M input by 1
LL_RCC_PLLSAI2M_DIV_2	PLLSAI2 division factor for PLLSAI2M input by 2
LL_RCC_PLLSAI2M_DIV_3	PLLSAI2 division factor for PLLSAI2M input by 3
LL_RCC_PLLSAI2M_DIV_4	PLLSAI2 division factor for PLLSAI2M input by 4
LL_RCC_PLLSAI2M_DIV_5	PLLSAI2 division factor for PLLSAI2M input by 5
LL_RCC_PLLSAI2M_DIV_6	PLLSAI2 division factor for PLLSAI2M input by 6
LL_RCC_PLLSAI2M_DIV_7	PLLSAI2 division factor for PLLSAI2M input by 7
LL_RCC_PLLSAI2M_DIV_8	PLLSAI2 division factor for PLLSAI2M input by 8
LL_RCC_PLLSAI2M_DIV_9	PLLSAI2 division factor for PLLSAI2M input by 9
LL_RCC_PLLSAI2M_DIV_10	PLLSAI2 division factor for PLLSAI2M input by 10
LL_RCC_PLLSAI2M_DIV_11	PLLSAI2 division factor for PLLSAI2M input by 11
LL_RCC_PLLSAI2M_DIV_12	PLLSAI2 division factor for PLLSAI2M input by 12
LL_RCC_PLLSAI2M_DIV_13	PLLSAI2 division factor for PLLSAI2M input by 13
LL_RCC_PLLSAI2M_DIV_14	PLLSAI2 division factor for PLLSAI2M input by 14

LL_RCC_PLLSAI2M_DIV_15	PLLSAI2 division factor for PLLSAI2M input by 15
LL_RCC_PLLSAI2M_DIV_16	PLLSAI2 division factor for PLLSAI2M input by 16

PLLSAI2 division factor (PLLSAI2P)

LL_RCC_PLLSAI2P_DIV_2	PLLSAI2 division factor for PLLSAI2P output by 2
LL_RCC_PLLSAI2P_DIV_3	PLLSAI2 division factor for PLLSAI2P output by 3
LL_RCC_PLLSAI2P_DIV_4	PLLSAI2 division factor for PLLSAI2P output by 4
LL_RCC_PLLSAI2P_DIV_5	PLLSAI2 division factor for PLLSAI2P output by 5
LL_RCC_PLLSAI2P_DIV_6	PLLSAI2 division factor for PLLSAI2P output by 6
LL_RCC_PLLSAI2P_DIV_7	PLLSAI2 division factor for PLLSAI2P output by 7
LL_RCC_PLLSAI2P_DIV_8	PLLSAI2 division factor for PLLSAI2P output by 8
LL_RCC_PLLSAI2P_DIV_9	PLLSAI2 division factor for PLLSAI2P output by 9
LL_RCC_PLLSAI2P_DIV_10	PLLSAI2 division factor for PLLSAI2P output by 10
LL_RCC_PLLSAI2P_DIV_11	PLLSAI2 division factor for PLLSAI2P output by 1
LL_RCC_PLLSAI2P_DIV_12	PLLSAI2 division factor for PLLSAI2P output by 12
LL_RCC_PLLSAI2P_DIV_13	PLLSAI2 division factor for PLLSAI2P output by 13
LL_RCC_PLLSAI2P_DIV_14	PLLSAI2 division factor for PLLSAI2P output by 14
LL_RCC_PLLSAI2P_DIV_15	PLLSAI2 division factor for PLLSAI2P output by 15
LL_RCC_PLLSAI2P_DIV_16	PLLSAI2 division factor for PLLSAI2P output by 16
LL_RCC_PLLSAI2P_DIV_17	PLLSAI2 division factor for PLLSAI2P output by 17
LL_RCC_PLLSAI2P_DIV_18	PLLSAI2 division factor for PLLSAI2P output by 18
LL_RCC_PLLSAI2P_DIV_19	PLLSAI2 division factor for PLLSAI2P output by 19
LL_RCC_PLLSAI2P_DIV_20	PLLSAI2 division factor for PLLSAI2P output by 20
LL_RCC_PLLSAI2P_DIV_21	PLLSAI2 division fctor for PLLSAI2P output by 21
LL_RCC_PLLSAI2P_DIV_22	PLLSAI2 division factor for PLLSAI2P output by 22
LL_RCC_PLLSAI2P_DIV_23	PLLSAI2 division factor for PLLSAI2P output by 23
LL_RCC_PLLSAI2P_DIV_24	PLLSAI2 division factor for PLLSAI2P output by 24
LL_RCC_PLLSAI2P_DIV_25	PLLSAI2 division factor for PLLSAI2P output by 25
LL_RCC_PLLSAI2P_DIV_26	PLLSAI2 division factor for PLLSAI2P output by 26
LL_RCC_PLLSAI2P_DIV_27	PLLSAI2 division factor for PLLSAI2P output by 27
LL_RCC_PLLSAI2P_DIV_28	PLLSAI2 division factor for PLLSAI2P output by 28
LL_RCC_PLLSAI2P_DIV_29	PLLSAI2 division factor for PLLSAI2P output by 29
LL_RCC_PLLSAI2P_DIV_30	PLLSAI2 division factor for PLLSAI2P output by 30
LL_RCC_PLLSAI2P_DIV_31	PLLSAI1 division factor for PLLSAI1P output by 31

PLLSAI2 division factor (PLLSAI2Q)

LL_RCC_PLLSAI2Q_DIV_2	PLLSAI2 division factor for PLLSAI2Q output by 2
LL_RCC_PLLSAI2Q_DIV_4	PLLSAI2 division factor for PLLSAI2Q output by 4

LL_RCC_PLLSAI2Q_DIV_6	PLLSAI2 division factor for PLLSAI2Q output by 6
LL_RCC_PLLSAI2Q_DIV_8	PLLSAI2 division factor for PLLSAI2Q output by 8

PLLSAI2 division factor (PLLSAI2R)

LL_RCC_PLLSAI2R_DIV_2	PLLSAI2 division factor for PLLSAI2R output by 2
LL_RCC_PLLSAI2R_DIV_4	PLLSAI2 division factor for PLLSAI2R output by 4
LL_RCC_PLLSAI2R_DIV_6	PLLSAI2 division factor for PLLSAI2R output by 6
LL_RCC_PLLSAI2R_DIV_8	PLLSAI2 division factor for PLLSAI2R output by 8

PLL, PLLSAI1 and PLLSAI2 entry clock source

LL_RCC_PLLSOURCE_NONE	No clock
LL_RCC_PLLSOURCE_MSI	MSI clock selected as PLL entry clock source
LL_RCC_PLLSOURCE_HSI	HSI16 clock selected as PLL entry clock source
LL_RCC_PLLSOURCE_HSE	HSE clock selected as PLL entry clock source

Peripheral RNG get clock source

LL_RCC_RNG_CLKSOURCE	RNG Clock source selection

Peripheral RNG clock source selection

LL_RCC_RNG_CLKSOURCE_HSI48	HSI48 clock used as RNG clock source
LL_RCC_RNG_CLKSOURCE_PLLSAI1	PLLSAI1 clock used as RNG clock source
LL_RCC_RNG_CLKSOURCE_PLL	PLL clock used as RNG clock source
LL_RCC_RNG_CLKSOURCE_MSI	MSI clock used as RNG clock source

RTC clock source selection

LL_RCC_RTC_CLKSOURCE_NONE	No clock used as RTC clock
LL_RCC_RTC_CLKSOURCE_LSE	LSE oscillator clock used as RTC clock
LL_RCC_RTC_CLKSOURCE_LSI	LSI oscillator clock used as RTC clock
LL_RCC_RTC_CLKSOURCE_HSE_DIV32	HSE oscillator clock divided by 32 used as RTC clock

Peripheral SAI get clock source

LL_RCC_SAI1_CLKSOURCE	SAI1 Clock source selection
LL_RCC_SAI2_CLKSOURCE	SAI2 Clock source selection

Peripheral SAI clock source selection

LL_RCC_SAI1_CLKSOURCE_PLL	PLL clock used as SAI1 clock source
LL_RCC_SAI1_CLKSOURCE_PLLSAI1	PLLSAI1 clock used as SAI1 clock source
LL_RCC_SAI1_CLKSOURCE_PLLSAI2	PLLSAI2 clock used as SAI1 clock source
LL_RCC_SAI1_CLKSOURCE_HSI	HSI clock used as SAI1 clock source
LL_RCC_SAI1_CLKSOURCE_PIN	External input clock used as SAI1 clock source
LL_RCC_SAI2_CLKSOURCE_PLL	PLL clock used as SAI2 clock source
LL_RCC_SAI2_CLKSOURCE_PLLSAI1	PLLSAI1 clock used as SAI2 clock source
LL_RCC_SAI2_CLKSOURCE_PLLSAI2	PLLSAI2 clock used as SAI2 clock source

LL_RCC_SAI2_CLKSOURCE_HSI	HSI clock used as SAI2 clock source
LL_RCC_SAI2_CLKSOURCE_PIN	External input clock used as SAI2 clock source

Peripheral SDMMC get clock source

LL_RCC_SDMMC1_CLKSOURCE	SDMMC1 Clock source selection

Peripheral SDMMC clock source selection

LL_RCC_SDMMC1_CLKSOURCE_48CLK	48MHz clock used as SDMMC1 clock source
LL_RCC_SDMMC1_CLKSOURCE_PLL	PLL clock used as SDMMC1 clock source

Wakeup from Stop and CSS backup clock selection

LL_RCC_STOP_WAKEUPCLOCK_MSI	MSI selection after wake-up from STOP
LL_RCC_STOP_WAKEUPCLOCK_HSI	HSI selection after wake-up from STOP

AHB prescaler

LL_RCC_SYSCLK_DIV_1	SYSCLK not divided
LL_RCC_SYSCLK_DIV_2	SYSCLK divided by 2
LL_RCC_SYSCLK_DIV_4	SYSCLK divided by 4
LL_RCC_SYSCLK_DIV_8	SYSCLK divided by 8
LL_RCC_SYSCLK_DIV_16	SYSCLK divided by 16
LL_RCC_SYSCLK_DIV_64	SYSCLK divided by 64
LL_RCC_SYSCLK_DIV_128	SYSCLK divided by 128
LL_RCC_SYSCLK_DIV_256	SYSCLK divided by 256
LL_RCC_SYSCLK_DIV_512	SYSCLK divided by 512

System clock switch

LL_RCC_SYS_CLKSOURCE_MSI	MSI selection as system clock
LL_RCC_SYS_CLKSOURCE_HSI	HSI selection as system clock
LL_RCC_SYS_CLKSOURCE_HSE	HSE selection as system clock
LL_RCC_SYS_CLKSOURCE_PLL	PLL selection as system clock

System clock switch status

LL_RCC_SYS_CLKSOURCE_STATUS_MSI	MSI used as system clock
LL_RCC_SYS_CLKSOURCE_STATUS_HSI	HSI used as system clock
LL_RCC_SYS_CLKSOURCE_STATUS_HSE	HSE used as system clock
LL_RCC_SYS_CLKSOURCE_STATUS_PLL	PLL used as system clock

Peripheral UART get clock source

LL_RCC_UART4_CLKSOURCE	UART4 Clock source selection
LL_RCC_UART5_CLKSOURCE	UART5 Clock source selection

Peripheral UART clock source selection

LL_RCC_UART4_CLKSOURCE_PCLK1	PCLK1 clock used as UART4 clock source
LL_RCC_UART4_CLKSOURCE_SYSCLK	SYSCLK clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_HSI	HSI clock used as UART4 clock source
LL_RCC_UART4_CLKSOURCE_LSE	LSE clock used as UART4 clock source
LL_RCC_UART5_CLKSOURCE_PCLK1	PCLK1 clock used as UART5 clock source
LL_RCC_UART5_CLKSOURCE_SYSCLK	SYSCLK clock used as UART5 clock source
LL_RCC_UART5_CLKSOURCE_HSI	HSI clock used as UART5 clock source
LL_RCC_UART5_CLKSOURCE_LSE	LSE clock used as UART5 clock source

Peripheral USART get clock source

LL_RCC_USART1_CLKSOURCE	USART1 Clock source selection
LL_RCC_USART2_CLKSOURCE	USART2 Clock source selection
LL_RCC_USART3_CLKSOURCE	USART3 Clock source selection

Peripheral USART clock source selection

LL_RCC_USART1_CLKSOURCE_PCLK2	PCLK2 clock used as USART1 clock source
LL_RCC_USART1_CLKSOURCE_SYSCLK	SYSCLK clock used as USART1 clock source
LL_RCC_USART1_CLKSOURCE_HSI	HSI clock used as USART1 clock source
LL_RCC_USART1_CLKSOURCE_LSE	LSE clock used as USART1 clock source
LL_RCC_USART2_CLKSOURCE_PCLK1	PCLK1 clock used as USART2 clock source
LL_RCC_USART2_CLKSOURCE_SYSCLK	SYSCLK clock used as USART2 clock source
LL_RCC_USART2_CLKSOURCE_HSI	HSI clock used as USART2 clock source
LL_RCC_USART2_CLKSOURCE_LSE	LSE clock used as USART2 clock source
LL_RCC_USART3_CLKSOURCE_PCLK1	PCLK1 clock used as USART3 clock source
LL_RCC_USART3_CLKSOURCE_SYSCLK	SYSCLK clock used as USART3 clock source
LL_RCC_USART3_CLKSOURCE_HSI	HSI clock used as USART3 clock source
LL_RCC_USART3_CLKSOURCE_LSE	LSE clock used as USART3 clock source

Peripheral USB get clock source

LL_RCC_USB_CLKSOURCE	USB Clock source selection

Peripheral USB clock source selection

LL_RCC_USB_CLKSOURCE_HSI48	HSI48 clock used as USB clock source
LL_RCC_USB_CLKSOURCE_PLLSAI1	PLLSAI1 clock used as USB clock source
LL_RCC_USB_CLKSOURCE_PLL	PLL clock used as USB clock source
LL_RCC_USB_CLKSOURCE_MSI	MSI clock used as USB clock source

Calculate frequencies

__LL_RCC_CALC_PLLCLK_FREQ

Description:

- Helper macro to calculate the PLLCLK frequency on system domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on MSI/HSE/HSI)
- __PLLM__: This parameter can be one of the following values:
  - LL_RCC_PLLM_DIV_1
  - LL_RCC_PLLM_DIV_2
  - LL_RCC_PLLM_DIV_3
  - LL_RCC_PLLM_DIV_4
  - LL_RCC_PLLM_DIV_5
  - LL_RCC_PLLM_DIV_6
  - LL_RCC_PLLM_DIV_7
  - LL_RCC_PLLM_DIV_8
  - LL_RCC_PLLM_DIV_9 (*)
  - LL_RCC_PLLM_DIV_10 (*)
  - LL_RCC_PLLM_DIV_11 (*)
  - LL_RCC_PLLM_DIV_12 (*)
  - LL_RCC_PLLM_DIV_13 (*)
  - LL_RCC_PLLM_DIV_14 (*)
  - LL_RCC_PLLM_DIV_15 (*)
  - LL_RCC_PLLM_DIV_16 (*)
- __PLLN__: Between 8 and 86
- __PLLR__: This parameter can be one of the following values:
  - LL_RCC_PLLR_DIV_2
  - LL_RCC_PLLR_DIV_4
  - LL_RCC_PLLR_DIV_6
  - LL_RCC_PLLR_DIV_8

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLCLK_FREQ (HSE_VALUE,LL_RCC_PLL_GetDivider (), LL_RCC_PLL_GetN (), LL_RCC_PLL_GetR ());

__LL_RCC_CALC_PLLCLK_ SAI_FREQ

Description:

- Helper macro to calculate the PLLCLK frequency used on SAI domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on MSI/HSE/HSI)
- __PLLM__: This parameter can be one of the following values:
  - LL_RCC_PLLM_DIV_1
  - LL_RCC_PLLM_DIV_2
  - LL_RCC_PLLM_DIV_3
  - LL_RCC_PLLM_DIV_4
  - LL_RCC_PLLM_DIV_5
  - LL_RCC_PLLM_DIV_6
  - LL_RCC_PLLM_DIV_7
  - LL_RCC_PLLM_DIV_8

- LL_RCC_PLLM_DIV_9 (*)
- LL_RCC_PLLM_DIV_10 (*)
- LL_RCC_PLLM_DIV_11 (*)
- LL_RCC_PLLM_DIV_12 (*)
- LL_RCC_PLLM_DIV_13 (*)
- LL_RCC_PLLM_DIV_14 (*)
- LL_RCC_PLLM_DIV_15 (*)
- LL_RCC_PLLM_DIV_16 (*)

- __PLLN__: Between 8 and 86
- __PLLP__: This parameter can be one of the following values:
  - LL_RCC_PLLP_DIV_2
  - LL_RCC_PLLP_DIV_3
  - LL_RCC_PLLP_DIV_4
  - LL_RCC_PLLP_DIV_5
  - LL_RCC_PLLP_DIV_6
  - LL_RCC_PLLP_DIV_7
  - LL_RCC_PLLP_DIV_8
  - LL_RCC_PLLP_DIV_9
  - LL_RCC_PLLP_DIV_10
  - LL_RCC_PLLP_DIV_11
  - LL_RCC_PLLP_DIV_12
  - LL_RCC_PLLP_DIV_13
  - LL_RCC_PLLP_DIV_14
  - LL_RCC_PLLP_DIV_15
  - LL_RCC_PLLP_DIV_16
  - LL_RCC_PLLP_DIV_17
  - LL_RCC_PLLP_DIV_18
  - LL_RCC_PLLP_DIV_19
  - LL_RCC_PLLP_DIV_20
  - LL_RCC_PLLP_DIV_21
  - LL_RCC_PLLP_DIV_22
  - LL_RCC_PLLP_DIV_23
  - LL_RCC_PLLP_DIV_24
  - LL_RCC_PLLP_DIV_25
  - LL_RCC_PLLP_DIV_26
  - LL_RCC_PLLP_DIV_27
  - LL_RCC_PLLP_DIV_28
  - LL_RCC_PLLP_DIV_29
  - LL_RCC_PLLP_DIV_30
  - LL_RCC_PLLP_DIV_31

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLCLK_SAI_FREQ (HSE_VALUE,LL_RCC_PLL_GetDivider (), LL_RCC_PLL_GetN (), LL_RCC_PLL_GetP ());

__LL_RCC_CALC_PLLCLK_48M_FREQ

Description:

- Helper macro to calculate the PLLCLK frequency used on 48M domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on MSI/HSE/HSI)
- __PLLM__: This parameter can be one of the following values:
  - LL_RCC_PLLM_DIV_1
  - LL_RCC_PLLM_DIV_2
  - LL_RCC_PLLM_DIV_3
  - LL_RCC_PLLM_DIV_4
  - LL_RCC_PLLM_DIV_5
  - LL_RCC_PLLM_DIV_6
  - LL_RCC_PLLM_DIV_7
  - LL_RCC_PLLM_DIV_8
  - LL_RCC_PLLM_DIV_9 (*)
  - LL_RCC_PLLM_DIV_10 (*)
  - LL_RCC_PLLM_DIV_11 (*)
  - LL_RCC_PLLM_DIV_12 (*)
  - LL_RCC_PLLM_DIV_13 (*)
  - LL_RCC_PLLM_DIV_14 (*)
  - LL_RCC_PLLM_DIV_15 (*)
  - LL_RCC_PLLM_DIV_16 (*)
- __PLLN__: Between 8 and 86
- __PLLQ__: This parameter can be one of the following values:
  - LL_RCC_PLLQ_DIV_2
  - LL_RCC_PLLQ_DIV_4
  - LL_RCC_PLLQ_DIV_6
  - LL_RCC_PLLQ_DIV_8

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLCLK_48M_FREQ (HSE_VALUE,LL_RCC_PLL_GetDivider (), LL_RCC_PLL_GetN (), LL_RCC_PLL_GetQ ());

__LL_RCC_CALC_PLLSAI1_SAI_FREQ

Description:

- Helper macro to calculate the PLLSAI1 frequency used for SAI domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on MSI/HSE/HSI)
- __PLLSAI1M__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI1M_DIV_1
  - LL_RCC_PLLSAI1M_DIV_2
  - LL_RCC_PLLSAI1M_DIV_3
  - LL_RCC_PLLSAI1M_DIV_4
  - LL_RCC_PLLSAI1M_DIV_5

- LL_RCC_PLLSAI1M_DIV_6
- LL_RCC_PLLSAI1M_DIV_7
- LL_RCC_PLLSAI1M_DIV_8
- LL_RCC_PLLSAI1M_DIV_9
- LL_RCC_PLLSAI1M_DIV_10
- LL_RCC_PLLSAI1M_DIV_11
- LL_RCC_PLLSAI1M_DIV_12
- LL_RCC_PLLSAI1M_DIV_13
- LL_RCC_PLLSAI1M_DIV_14
- LL_RCC_PLLSAI1M_DIV_15
- LL_RCC_PLLSAI1M_DIV_16

- __PLLSAI1N__: Between 8 and 86
- __PLLSAI1P__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI1P_DIV_2
  - LL_RCC_PLLSAI1P_DIV_3
  - LL_RCC_PLLSAI1P_DIV_4
  - LL_RCC_PLLSAI1P_DIV_5
  - LL_RCC_PLLSAI1P_DIV_6
  - LL_RCC_PLLSAI1P_DIV_7
  - LL_RCC_PLLSAI1P_DIV_8
  - LL_RCC_PLLSAI1P_DIV_9
  - LL_RCC_PLLSAI1P_DIV_10
  - LL_RCC_PLLSAI1P_DIV_11
  - LL_RCC_PLLSAI1P_DIV_12
  - LL_RCC_PLLSAI1P_DIV_13
  - LL_RCC_PLLSAI1P_DIV_14
  - LL_RCC_PLLSAI1P_DIV_15
  - LL_RCC_PLLSAI1P_DIV_16
  - LL_RCC_PLLSAI1P_DIV_17
  - LL_RCC_PLLSAI1P_DIV_18
  - LL_RCC_PLLSAI1P_DIV_19
  - LL_RCC_PLLSAI1P_DIV_20
  - LL_RCC_PLLSAI1P_DIV_21
  - LL_RCC_PLLSAI1P_DIV_22
  - LL_RCC_PLLSAI1P_DIV_23
  - LL_RCC_PLLSAI1P_DIV_24
  - LL_RCC_PLLSAI1P_DIV_25
  - LL_RCC_PLLSAI1P_DIV_26
  - LL_RCC_PLLSAI1P_DIV_27
  - LL_RCC_PLLSAI1P_DIV_28
  - LL_RCC_PLLSAI1P_DIV_29
  - LL_RCC_PLLSAI1P_DIV_30
  - LL_RCC_PLLSAI1P_DIV_31

Return value:

- PLLSAI1: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLSAI1_SAI_FREQ (HSE_VALUE,LL_RCC_PLLSAI1_GetDivider (), LL_RCC_PLLSAI1_GetN (), LL_RCC_PLLSAI1_GetP

());

__LL_RCC_CALC_PLLSAI1_48M_FREQ

Description:

- Helper macro to calculate the PLLSAI1 frequency used on 48M domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on MSI/HSE/HSI)
- __PLLSAI1M__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI1M_DIV_1
  - LL_RCC_PLLSAI1M_DIV_2
  - LL_RCC_PLLSAI1M_DIV_3
  - LL_RCC_PLLSAI1M_DIV_4
  - LL_RCC_PLLSAI1M_DIV_5
  - LL_RCC_PLLSAI1M_DIV_6
  - LL_RCC_PLLSAI1M_DIV_7
  - LL_RCC_PLLSAI1M_DIV_8
  - LL_RCC_PLLSAI1M_DIV_9
  - LL_RCC_PLLSAI1M_DIV_10
  - LL_RCC_PLLSAI1M_DIV_11
  - LL_RCC_PLLSAI1M_DIV_12
  - LL_RCC_PLLSAI1M_DIV_13
  - LL_RCC_PLLSAI1M_DIV_14
  - LL_RCC_PLLSAI1M_DIV_15
  - LL_RCC_PLLSAI1M_DIV_16
- __PLLSAI1N__: Between 8 and 86
- __PLLSAI1Q__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI1Q_DIV_2
  - LL_RCC_PLLSAI1Q_DIV_4
  - LL_RCC_PLLSAI1Q_DIV_6
  - LL_RCC_PLLSAI1Q_DIV_8

Return value:

- PLLSAI1: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLSAI1_48M_FREQ (HSE_VALUE,LL_RCC_PLLSAI1_GetDivider (), LL_RCC_PLLSAI1_GetN (), LL_RCC_PLLSAI1_GetQ ());

__LL_RCC_CALC_PLLSAI1_ADC_FREQ

Description:

- Helper macro to calculate the PLLSAI1 frequency used on ADC domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on MSI/HSE/HSI)
- __PLLSAI1M__: This parameter can be one of the following values:

- LL_RCC_PLLSAI1M_DIV_1
- LL_RCC_PLLSAI1M_DIV_2
- LL_RCC_PLLSAI1M_DIV_3
- LL_RCC_PLLSAI1M_DIV_4
- LL_RCC_PLLSAI1M_DIV_5
- LL_RCC_PLLSAI1M_DIV_6
- LL_RCC_PLLSAI1M_DIV_7
- LL_RCC_PLLSAI1M_DIV_8
- LL_RCC_PLLSAI1M_DIV_9
- LL_RCC_PLLSAI1M_DIV_10
- LL_RCC_PLLSAI1M_DIV_11
- LL_RCC_PLLSAI1M_DIV_12
- LL_RCC_PLLSAI1M_DIV_13
- LL_RCC_PLLSAI1M_DIV_14
- LL_RCC_PLLSAI1M_DIV_15
- LL_RCC_PLLSAI1M_DIV_16

- __PLLSAI1N__: Between 8 and 86
- __PLLSAI1R__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI1R_DIV_2
  - LL_RCC_PLLSAI1R_DIV_4
  - LL_RCC_PLLSAI1R_DIV_6
  - LL_RCC_PLLSAI1R_DIV_8

Return value:

- PLLSAI1: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLSAI1_ADC_FREQ (HSE_VALUE,LL_RCC_PLLSAI1_GetDivider (), LL_RCC_PLLSAI1_GetN (), LL_RCC_PLLSAI1_GetR ());

__LL_RCC_CALC_PLLSAI2_SAI_FREQ

Description:

- Helper macro to calculate the PLLSAI2 frequency used for SAI domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on MSI/HSE/HSI)
- __PLLSAI2M__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI2M_DIV_1
  - LL_RCC_PLLSAI2M_DIV_2
  - LL_RCC_PLLSAI2M_DIV_3
  - LL_RCC_PLLSAI2M_DIV_4
  - LL_RCC_PLLSAI2M_DIV_5
  - LL_RCC_PLLSAI2M_DIV_6
  - LL_RCC_PLLSAI2M_DIV_7
  - LL_RCC_PLLSAI2M_DIV_8
  - LL_RCC_PLLSAI2M_DIV_9
  - LL_RCC_PLLSAI2M_DIV_10
  - LL_RCC_PLLSAI2M_DIV_11

- LL_RCC_PLLSAI2M_DIV_12
- LL_RCC_PLLSAI2M_DIV_13
- LL_RCC_PLLSAI2M_DIV_14
- LL_RCC_PLLSAI2M_DIV_15
- LL_RCC_PLLSAI2M_DIV_16

- __PLLSAI2N__: Between 8 and 86
- __PLLSAI2P__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI2P_DIV_2
  - LL_RCC_PLLSAI2P_DIV_3
  - LL_RCC_PLLSAI2P_DIV_4
  - LL_RCC_PLLSAI2P_DIV_5
  - LL_RCC_PLLSAI2P_DIV_6
  - LL_RCC_PLLSAI2P_DIV_7
  - LL_RCC_PLLSAI2P_DIV_8
  - LL_RCC_PLLSAI2P_DIV_9
  - LL_RCC_PLLSAI2P_DIV_10
  - LL_RCC_PLLSAI2P_DIV_11
  - LL_RCC_PLLSAI2P_DIV_12
  - LL_RCC_PLLSAI2P_DIV_13
  - LL_RCC_PLLSAI2P_DIV_14
  - LL_RCC_PLLSAI2P_DIV_15
  - LL_RCC_PLLSAI2P_DIV_16
  - LL_RCC_PLLSAI2P_DIV_17
  - LL_RCC_PLLSAI2P_DIV_18
  - LL_RCC_PLLSAI2P_DIV_19
  - LL_RCC_PLLSAI2P_DIV_20
  - LL_RCC_PLLSAI2P_DIV_21
  - LL_RCC_PLLSAI2P_DIV_22
  - LL_RCC_PLLSAI2P_DIV_23
  - LL_RCC_PLLSAI2P_DIV_24
  - LL_RCC_PLLSAI2P_DIV_25
  - LL_RCC_PLLSAI2P_DIV_26
  - LL_RCC_PLLSAI2P_DIV_27
  - LL_RCC_PLLSAI2P_DIV_28
  - LL_RCC_PLLSAI2P_DIV_29
  - LL_RCC_PLLSAI2P_DIV_30
  - LL_RCC_PLLSAI2P_DIV_31

Return value:

- PLLSAI2: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLSAI2_SAI_FREQ (HSE_VALUE,LL_RCC_PLLSAI2_GetDivider (), LL_RCC_PLLSAI2_GetN (), LL_RCC_PLLSAI2_GetP ());

__LL_RCC_CALC_PLLSAI2_LTDC_FREQ

Description:

- Helper macro to calculate the PLLSAI2 frequency used for LTDC domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on HSE/HSI/MSI)
- __PLLSAI2M__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI2M_DIV_1
  - LL_RCC_PLLSAI2M_DIV_2
  - LL_RCC_PLLSAI2M_DIV_3
  - LL_RCC_PLLSAI2M_DIV_4
  - LL_RCC_PLLSAI2M_DIV_5
  - LL_RCC_PLLSAI2M_DIV_6
  - LL_RCC_PLLSAI2M_DIV_7
  - LL_RCC_PLLSAI2M_DIV_8
  - LL_RCC_PLLSAI2M_DIV_9
  - LL_RCC_PLLSAI2M_DIV_10
  - LL_RCC_PLLSAI2M_DIV_11
  - LL_RCC_PLLSAI2M_DIV_12
  - LL_RCC_PLLSAI2M_DIV_13
  - LL_RCC_PLLSAI2M_DIV_14
  - LL_RCC_PLLSAI2M_DIV_15
  - LL_RCC_PLLSAI2M_DIV_16
- __PLLSAI2N__: Between 8 and 86
- __PLLSAI2R__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI2R_DIV_2
  - LL_RCC_PLLSAI2R_DIV_4
  - LL_RCC_PLLSAI2R_DIV_6
  - LL_RCC_PLLSAI2R_DIV_8
- __PLLSAI2DIVR__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI2DIVR_DIV_2
  - LL_RCC_PLLSAI2DIVR_DIV_4
  - LL_RCC_PLLSAI2DIVR_DIV_8
  - LL_RCC_PLLSAI2DIVR_DIV_16

Return value:

- PLLSAI2: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLSAI2_LTDC_FREQ (HSE_VALUE,LL_RCC_PLLSAI2_GetDivider (), LL_RCC_PLLSAI2_GetN (), LL_RCC_PLLSAI2_GetR (), LL_RCC_PLLSAI2_GetDIVR ());

__LL_RCC_CALC_PLLSAI2_DSI_FREQ

Description:

- Helper macro to calculate the PLLDSICLK frequency used on DSI.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on HSE/HSI/MSI)
- __PLLSAI2M__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI2M_DIV_1

- LL_RCC_PLLSAI2M_DIV_2
- LL_RCC_PLLSAI2M_DIV_3
- LL_RCC_PLLSAI2M_DIV_4
- LL_RCC_PLLSAI2M_DIV_5
- LL_RCC_PLLSAI2M_DIV_6
- LL_RCC_PLLSAI2M_DIV_7
- LL_RCC_PLLSAI2M_DIV_8
- LL_RCC_PLLSAI2M_DIV_9
- LL_RCC_PLLSAI2M_DIV_10
- LL_RCC_PLLSAI2M_DIV_11
- LL_RCC_PLLSAI2M_DIV_12
- LL_RCC_PLLSAI2M_DIV_13
- LL_RCC_PLLSAI2M_DIV_14
- LL_RCC_PLLSAI2M_DIV_15
- LL_RCC_PLLSAI2M_DIV_16

- __PLLSAI2N__: Between 8 and 86
- __PLLSAI2Q__: This parameter can be one of the following values:
  - LL_RCC_PLLSAI2Q_DIV_2
  - LL_RCC_PLLSAI2Q_DIV_4
  - LL_RCC_PLLSAI2Q_DIV_6
  - LL_RCC_PLLSAI2Q_DIV_8

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLSAI2_DSI_FREQ (HSE_VALUE,LL_RCC_PLLSAI2_GetDivider (), LL_RCC_PLLSAI2_GetN (), LL_RCC_PLLSAI2_GetQ ());

__LL_RCC_CALC_HCLK_FREQ

Description:

- Helper macro to calculate the HCLK frequency.

Parameters:

- __SYSCLKFREQ__: SYSCLK frequency (based on MSI/HSE/HSI/PLLCLK)
- __AHBPRESCALER__: This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return value:

- HCLK: clock frequency (in Hz)

__LL_RCC_CALC_PCLK1_FREQ

Description:

- Helper macro to calculate the PCLK1 frequency (ABP1)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB1PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

__LL_RCC_CALC_PCLK2_FREQ

Description:

- Helper macro to calculate the PCLK2 frequency (ABP2)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB2PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return value:

- PCLK2: clock frequency (in Hz)

__LL_RCC_CALC_MSI_FREQ

Description:

- Helper macro to calculate the MSI frequency (in Hz)

Parameters:

- __MSISEL__: This parameter can be one of the following values:
  - LL_RCC_MSIRANGESEL_STANDBY
  - LL_RCC_MSIRANGESEL_RUN
- __MSIRANGE__: This parameter can be one of the following values:
  - LL_RCC_MSIRANGE_0
  - LL_RCC_MSIRANGE_1
  - LL_RCC_MSIRANGE_2
  - LL_RCC_MSIRANGE_3
  - LL_RCC_MSIRANGE_4
  - LL_RCC_MSIRANGE_5
  - LL_RCC_MSIRANGE_6
  - LL_RCC_MSIRANGE_7
  - LL_RCC_MSIRANGE_8

  - LL_RCC_MSIRANGE_9
  - LL_RCC_MSIRANGE_10
  - LL_RCC_MSIRANGE_11
  - LL_RCC_MSISRANGE_4
  - LL_RCC_MSISRANGE_5
  - LL_RCC_MSISRANGE_6
  - LL_RCC_MSISRANGE_7

Return value:

- MSI: clock frequency (in Hz)

Notes:

- __MSISEL__ can be retrieved thanks to function LL_RCC_MSI_IsEnabledRangeSelect() if __MSISEL__ is equal to LL_RCC_MSIRANGESEL_STANDBY, __MSIRANGE__can be retrieved by LL_RCC_MSI_GetRangeAfterStandby() else by LL_RCC_MSI_GetRange() ex: __LL_RCC_CALC_MSI_FREQ(LL_RCC_MSI_IsEnabledRangeSelect(), (LL_RCC_MSI_IsEnabledRangeSelect()? LL_RCC_MSI_GetRange(): LL_RCC_MSI_GetRangeAfterStandby()))

Common Write and read registers Macros

LL_RCC_WriteReg

Description:

- Write a value in RCC register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RCC_ReadReg

Description:

- Read a value in RCC register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

92 LL RNG Generic Driver

92.1 RNG Firmware driver registers structures

92.1.1 LL_RNG_InitTypeDef

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t LL_RNG_InitTypeDef::ClockErrorDetection***
  Clock error detection. This parameter can be one value of ***RNG_LL_CED***.This parameter can be modified using unitary functions **LL_RNG_EnableClkErrorDetect()**.

92.2 RNG Firmware driver API description

92.2.1 Detailed description of functions

LL_RNG_Enable

Function name	**__STATIC_INLINE void LL_RNG_Enable (RNG_TypeDef * RNGx)**
Function description	Enable Random Number Generation.
Parameters	• **RNGx:** RNG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR RNGEN LL_RNG_Enable

LL_RNG_Disable

Function name	**__STATIC_INLINE void LL_RNG_Disable (RNG_TypeDef * RNGx)**
Function description	Disable Random Number Generation.
Parameters	• **RNGx:** RNG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR RNGEN LL_RNG_Disable

LL_RNG_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsEnabled (RNG_TypeDef * RNGx)**
Function description	Check if Random Number Generator is enabled.
Parameters	• **RNGx:** RNG Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR RNGEN LL_RNG_IsEnabled

LL_RNG_EnableClkErrorDetect

Function name	**__STATIC_INLINE void LL_RNG_EnableClkErrorDetect (RNG_TypeDef * RNGx)**
Function description	Enable RNG Clock Error Detection.
Parameters	• **RNGx:** RNG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CED LL_RNG_EnableClkErrorDetect

LL_RNG_DisableClkErrorDetect

Function name	**__STATIC_INLINE void LL_RNG_DisableClkErrorDetect (RNG_TypeDef * RNGx)**
Function description	Disable RNG Clock Error Detection.
Parameters	• **RNGx:** RNG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CED LL_RNG_DisableClkErrorDetect

LL_RNG_IsEnabledClkErrorDetect

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsEnabledClkErrorDetect (RNG_TypeDef * RNGx)**
Function description	Check if RNG Clock Error Detection is enabled.
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR CED LL_RNG_IsEnabledClkErrorDetect

LL_RNG_IsActiveFlag_DRDY

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_DRDY (RNG_TypeDef * RNGx)**
Function description	Indicate if the RNG Data ready Flag is set or not.
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• SR DRDY LL_RNG_IsActiveFlag_DRDY

LL_RNG_IsActiveFlag_CECS

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CECS (RNG_TypeDef * RNGx)**
Function description	Indicate if the Clock Error Current Status Flag is set or not.
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CECS LL_RNG_IsActiveFlag_CECS

LL_RNG_IsActiveFlag_SECS

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SECS (RNG_TypeDef * RNGx)**
Function description	Indicate if the Seed Error Current Status Flag is set or not.
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR SECS LL_RNG_IsActiveFlag_SECS

LL_RNG_IsActiveFlag_CEIS

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CEIS (RNG_TypeDef * RNGx)**
Function description	Indicate if the Clock Error Interrupt Status Flag is set or not.
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CEIS LL_RNG_IsActiveFlag_CEIS

LL_RNG_IsActiveFlag_SEIS

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SEIS (RNG_TypeDef * RNGx)**
Function description	Indicate if the Seed Error Interrupt Status Flag is set or not.
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• SR SEIS LL_RNG_IsActiveFlag_SEIS

reference:

LL_RNG_ClearFlag_CEIS

Function name	**__STATIC_INLINE void LL_RNG_ClearFlag_CEIS (RNG_TypeDef * RNGx)**
Function description	Clear Clock Error interrupt Status (CEIS) Flag.
Parameters	• **RNGx:** RNG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CEIS LL_RNG_ClearFlag_CEIS

LL_RNG_ClearFlag_SEIS

Function name	**__STATIC_INLINE void LL_RNG_ClearFlag_SEIS (RNG_TypeDef * RNGx)**
Function description	Clear Seed Error interrupt Status (SEIS) Flag.
Parameters	• **RNGx:** RNG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR SEIS LL_RNG_ClearFlag_SEIS

LL_RNG_EnableIT

Function name	**__STATIC_INLINE void LL_RNG_EnableIT (RNG_TypeDef * RNGx)**
Function description	Enable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)
Parameters	• **RNGx:** RNG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR IE LL_RNG_EnableIT

LL_RNG_DisableIT

Function name	**__STATIC_INLINE void LL_RNG_DisableIT (RNG_TypeDef * RNGx)**
Function description	Disable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)
Parameters	• **RNGx:** RNG Instance
Return values	• **None:**
Reference Manual to LL API cross	• CR IE LL_RNG_DisableIT

reference:

LL_RNG_IsEnabledIT

Function name	**__STATIC_INLINE uint32_t LL_RNG_IsEnabledIT (RNG_TypeDef * RNGx)**
Function description	Check if Random Number Generator Interrupt is enabled (applies for either Seed error, Clock Error or Data ready interrupts)
Parameters	• **RNGx:** RNG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR IE LL_RNG_IsEnabledIT

LL_RNG_ReadRandData32

Function name	**__STATIC_INLINE uint32_t LL_RNG_ReadRandData32 (RNG_TypeDef * RNGx)**
Function description	Return32-bit Random Number value.
Parameters	• **RNGx:** RNG Instance
Return values	• **Generated:** 32-bit random value
Reference Manual to LL API cross reference:	• DR RNDATA LL_RNG_ReadRandData32

LL_RNG_Init

Function name	**ErrorStatus LL_RNG_Init (RNG_TypeDef * RNGx, LL_RNG_InitTypeDef * RNG_InitStruct)**
Function description	Initialize RNG registers according to the specified parameters in RNG_InitStruct.
Parameters	• **RNGx:** RNG Instance • **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure that contains the configuration information for the specified RNG peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RNG registers are initialized according to RNG_InitStruct content – ERROR: not applicable

LL_RNG_DeInit

Function name	**ErrorStatus LL_RNG_DeInit (RNG_TypeDef * RNGx)**
Function description	De-initialize RNG registers (Registers restored to their default values).
Parameters	• **RNGx:** RNG Instance

Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RNG registers are de-initialized – ERROR: not applicable

92.3 RNG Firmware driver defines

92.3.1 RNG

Clock Error Detection

LL_RNG_CED_ENABLE Clock error detection enabled

LL_RNG_CED_DISABLE Clock error detection disabled

Get Flags Defines

LL_RNG_SR_DRDY Register contains valid random data

LL_RNG_SR_CECS Clock error current status

LL_RNG_SR_SECS Seed error current status

LL_RNG_SR_CEIS Clock error interrupt status

LL_RNG_SR_SEIS Seed error interrupt status

IT Defines

LL_RNG_CR_IE RNG Interrupt enable

Common Write and read registers Macros

LL_RNG_WriteReg **Description:**

- Write a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RNG_ReadReg **Description:**

- Read a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be read

Return value:

- Register: value

93 LL RTC Generic Driver

93.1 RTC Firmware driver registers structures

93.1.1 LL_RTC_InitTypeDef

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrescaler***
- ***uint32_t SynchPrescaler***

Field Documentation

- ***uint32_t LL_RTC_InitTypeDef::HourFormat***
  Specifies the RTC Hours Format. This parameter can be a value of ***RTC_LL_EC_HOURFORMAT***This feature can be modified afterwards using unitary function **LL_RTC_SetHourFormat()**.
- ***uint32_t LL_RTC_InitTypeDef::AsynchPrescaler***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FThis feature can be modified afterwards using unitary function **LL_RTC_SetAsynchPrescaler()**.
- ***uint32_t LL_RTC_InitTypeDef::SynchPrescaler***
  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFFThis feature can be modified afterwards using unitary function **LL_RTC_SetSynchPrescaler()**.

93.1.2 LL_RTC_TimeTypeDef

Data Fields

- ***uint32_t TimeFormat***
- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

Field Documentation

- ***uint32_t LL_RTC_TimeTypeDef::TimeFormat***
  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_LL_EC_TIME_FORMAT***This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetFormat()**.
- ***uint8_t LL_RTC_TimeTypeDef::Hours***
  Specifies the RTC Time Hours. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the **LL_RTC_TIME_FORMAT_PM** is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the **LL_RTC_TIME_FORMAT_AM_OR_24** is selected.This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetHour()**.
- ***uint8_t LL_RTC_TimeTypeDef::Minutes***
  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetMinute()**.
- ***uint8_t LL_RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between

Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetSecond()**.

93.1.3 LL_RTC_DateTypeDef

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Day***
- ***uint8_t Year***

Field Documentation

- ***uint8_t LL_RTC_DateTypeDef::WeekDay***
  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_LL_EC_WEEKDAY***This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetWeekDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Month***
  Specifies the RTC Date Month. This parameter can be a value of ***RTC_LL_EC_MONTH***This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetMonth()**.
- ***uint8_t LL_RTC_DateTypeDef::Day***
  Specifies the RTC Date Day. This parameter must be a number between Min_Data = 1 and Max_Data = 31This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Year***
  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetYear()**.

93.1.4 LL_RTC_AlarmTypeDef

Data Fields

- ***LL_RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***

Field Documentation

- ***LL_RTC_TimeTypeDef LL_RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members.
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_LL_EC_ALMA_MASK*** for ALARM A or ***RTC_LL_EC_ALMB_MASK*** for ALARM B.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetMask()** for ALARM A or **LL_RTC_ALMB_SetMask()** for ALARM B
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on day or WeekDay. This parameter can be a value of ***RTC_LL_EC_ALMA_WEEKDAY_SELECTION*** for ALARM A or ***RTC_LL_EC_ALMB_WEEKDAY_SELECTION*** for ALARM BThis feature can be modified afterwards using unitary function **LL_RTC_ALMA_EnableWeekday()** or **LL_RTC_ALMA_DisableWeekday()** for ALARM A or **LL_RTC_ALMB_EnableWeekday()** or **LL_RTC_ALMB_DisableWeekday()** for ALARM B

- ***uint8_t LL_RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Day/WeekDay. If AlarmDateWeekDaySel set to day, this parameter must be a number between Min_Data = 1 and Max_Data = 31.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetDay()** for ALARM A or **LL_RTC_ALMB_SetDay()** for ALARM B.If AlarmDateWeekDaySel set to Weekday, this parameter can be a value of ***RTC_LL_EC_WEEKDAY***.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetWeekDay()** for ALARM A or **LL_RTC_ALMB_SetWeekDay()** for ALARM B.

93.2 RTC Firmware driver API description

93.2.1 Detailed description of functions

LL_RTC_SetHourFormat

Function name	**__STATIC_INLINE void LL_RTC_SetHourFormat (RTC_TypeDef * RTCx, uint32_t HourFormat)**
Function description	Set Hours format (24 hour/day or AM/PM hour format)
Parameters	• **RTCx:** RTC Instance • **HourFormat:** This parameter can be one of the following values: – LL_RTC_HOURFORMAT_24HOUR – LL_RTC_HOURFORMAT_AMPM
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function)
Reference Manual to LL API cross reference:	• CR FMT LL_RTC_SetHourFormat

LL_RTC_GetHourFormat

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetHourFormat (RTC_TypeDef * RTCx)**
Function description	Get Hours format (24 hour/day or AM/PM hour format)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_HOURFORMAT_24HOUR – LL_RTC_HOURFORMAT_AMPM
Reference Manual to LL API cross reference:	• CR FMT LL_RTC_GetHourFormat

LL_RTC_SetAlarmOutEvent

Function name	**__STATIC_INLINE void LL_RTC_SetAlarmOutEvent (RTC_TypeDef * RTCx, uint32_t AlarmOutput)**

Function description	Select the flag to be routed to RTC_ALARM output.
Parameters	• **RTCx:** RTC Instance • **AlarmOutput:** This parameter can be one of the following values: – LL_RTC_ALARMOUT_DISABLE – LL_RTC_ALARMOUT_ALMA – LL_RTC_ALARMOUT_ALMB – LL_RTC_ALARMOUT_WAKEUP
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR OSEL LL_RTC_SetAlarmOutEvent

LL_RTC_GetAlarmOutEvent

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutEvent (RTC_TypeDef * RTCx)**
Function description	Get the flag to be routed to RTC_ALARM output.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_ALARMOUT_DISABLE – LL_RTC_ALARMOUT_ALMA – LL_RTC_ALARMOUT_ALMB – LL_RTC_ALARMOUT_WAKEUP
Reference Manual to LL API cross reference:	• CR OSEL LL_RTC_GetAlarmOutEvent

LL_RTC_SetAlarmOutputType

Function name	**__STATIC_INLINE void LL_RTC_SetAlarmOutputType (RTC_TypeDef * RTCx, uint32_t Output)**
Function description	Set RTC_ALARM output type (ALARM in push-pull or open-drain output)
Parameters	• **RTCx:** RTC Instance • **Output:** This parameter can be one of the following values: – LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN – LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL
Return values	• **None:**
Notes	• Used only when RTC_ALARM is mapped on PC13
Reference Manual to LL API cross reference:	• OR ALARMOUTTYPE LL_RTC_SetAlarmOutputType

LL_RTC_GetAlarmOutputType

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutputType (RTC_TypeDef * RTCx)**
Function description	Get RTC_ALARM output type (ALARM in push-pull or open-drain output)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN – LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL
Notes	• used only when RTC_ALARM is mapped on PC13
Reference Manual to LL API cross reference:	• OR ALARMOUTTYPE LL_RTC_GetAlarmOutputType

LL_RTC_EnableInitMode

Function name	**__STATIC_INLINE void LL_RTC_EnableInitMode (RTC_TypeDef * RTCx)**
Function description	Enable initialization mode.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Initialization mode is used to program time and date register (RTC_TR and RTC_DR) and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.
Reference Manual to LL API cross reference:	• ISR INIT LL_RTC_EnableInitMode

LL_RTC_DisableInitMode

Function name	**__STATIC_INLINE void LL_RTC_DisableInitMode (RTC_TypeDef * RTCx)**
Function description	Disable initialization mode (Free running mode)
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR INIT LL_RTC_DisableInitMode

LL_RTC_SetOutputPolarity

Function name	**__STATIC_INLINE void LL_RTC_SetOutputPolarity (RTC_TypeDef * RTCx, uint32_t Polarity)**
Function description	Set Output polarity (pin is low when ALRAF/ALRBF/WUTF is

	asserted)
Parameters	• **RTCx:** RTC Instance • **Polarity:** This parameter can be one of the following values: – LL_RTC_OUTPUTPOLARITY_PIN_HIGH – LL_RTC_OUTPUTPOLARITY_PIN_LOW
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR POL LL_RTC_SetOutputPolarity

LL_RTC_GetOutputPolarity

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetOutputPolarity (RTC_TypeDef * RTCx)**
Function description	Get Output polarity.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_OUTPUTPOLARITY_PIN_HIGH – LL_RTC_OUTPUTPOLARITY_PIN_LOW
Reference Manual to LL API cross reference:	• CR POL LL_RTC_GetOutputPolarity

LL_RTC_EnableShadowRegBypass

Function name	**__STATIC_INLINE void LL_RTC_EnableShadowRegBypass (RTC_TypeDef * RTCx)**
Function description	Enable Bypass the shadow registers.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR BYPSHAD LL_RTC_EnableShadowRegBypass

LL_RTC_DisableShadowRegBypass

Function name	**__STATIC_INLINE void LL_RTC_DisableShadowRegBypass (RTC_TypeDef * RTCx)**
Function description	Disable Bypass the shadow registers.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• CR BYPSHAD LL_RTC_DisableShadowRegBypass

LL_RTC_IsShadowRegBypassEnabled

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsShadowRegBypassEnabled (RTC_TypeDef * RTCx)**
Function description	Check if Shadow registers bypass is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR BYPSHAD LL_RTC_IsShadowRegBypassEnabled

LL_RTC_EnableRefClock

Function name	**__STATIC_INLINE void LL_RTC_EnableRefClock (RTC_TypeDef * RTCx)**
Function description	Enable RTC_REFIN reference clock detection (50 or 60 Hz)
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function)
Reference Manual to LL API cross reference:	• CR REFCKON LL_RTC_EnableRefClock

LL_RTC_DisableRefClock

Function name	**__STATIC_INLINE void LL_RTC_DisableRefClock (RTC_TypeDef * RTCx)**
Function description	Disable RTC_REFIN reference clock detection (50 or 60 Hz)
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function)
Reference Manual to LL API cross reference:	• CR REFCKON LL_RTC_DisableRefClock

LL_RTC_SetAsynchPrescaler

Function name	**__STATIC_INLINE void LL_RTC_SetAsynchPrescaler (RTC_TypeDef * RTCx, uint32_t AsynchPrescaler)**
Function description	Set Asynchronous prescaler factor.
Parameters	• **RTCx:** RTC Instance • **AsynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7F
Return values	• **None:**
Reference Manual to LL API cross reference:	• PRER PREDIV_A LL_RTC_SetAsynchPrescaler

LL_RTC_SetSynchPrescaler

Function name	**__STATIC_INLINE void LL_RTC_SetSynchPrescaler (RTC_TypeDef * RTCx, uint32_t SynchPrescaler)**
Function description	Set Synchronous prescaler factor.
Parameters	• **RTCx:** RTC Instance • **SynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7FFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• PRER PREDIV_S LL_RTC_SetSynchPrescaler

LL_RTC_GetAsynchPrescaler

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetAsynchPrescaler (RTC_TypeDef * RTCx)**
Function description	Get Asynchronous prescaler factor.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data = 0 and Max_Data = 0x7F
Reference Manual to LL API cross reference:	• PRER PREDIV_A LL_RTC_GetAsynchPrescaler

LL_RTC_GetSynchPrescaler

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetSynchPrescaler (RTC_TypeDef * RTCx)**
Function description	Get Synchronous prescaler factor.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data = 0 and Max_Data = 0x7FFF
Reference Manual to LL API cross	• PRER PREDIV_S LL_RTC_GetSynchPrescaler

reference:

LL_RTC_EnableWriteProtection

Function name	**__STATIC_INLINE void LL_RTC_EnableWriteProtection (RTC_TypeDef * RTCx)**
Function description	Enable the write protection for RTC registers.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• WPR KEY LL_RTC_EnableWriteProtection

LL_RTC_DisableWriteProtection

Function name	**__STATIC_INLINE void LL_RTC_DisableWriteProtection (RTC_TypeDef * RTCx)**
Function description	Disable the write protection for RTC registers.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• WPR KEY LL_RTC_DisableWriteProtection

LL_RTC_EnableOutRemap

Function name	**__STATIC_INLINE void LL_RTC_EnableOutRemap (RTC_TypeDef * RTCx)**
Function description	Enable RTC_OUT remap.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• OR OUT_RMP LL_RTC_EnableOutRemap

LL_RTC_DisableOutRemap

Function name	**__STATIC_INLINE void LL_RTC_DisableOutRemap (RTC_TypeDef * RTCx)**
Function description	Disable RTC_OUT remap.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• OR OUT_RMP LL_RTC_DisableOutRemap

LL_RTC_TIME_SetFormat

Function name	**__STATIC_INLINE void LL_RTC_TIME_SetFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)**
Function description	Set time format (AM/24-hour or PM notation)
Parameters	• **RTCx:** RTC Instance • **TimeFormat:** This parameter can be one of the following values: – LL_RTC_TIME_FORMAT_AM_OR_24 – LL_RTC_TIME_FORMAT_PM
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function)
Reference Manual to LL API cross reference:	• TR PM LL_RTC_TIME_SetFormat

LL_RTC_TIME_GetFormat

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_GetFormat (RTC_TypeDef * RTCx)**
Function description	Get time format (AM or PM notation)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_TIME_FORMAT_AM_OR_24 – LL_RTC_TIME_FORMAT_PM
Notes	• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit • Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
Reference Manual to LL API cross reference:	• TR PM LL_RTC_TIME_GetFormat

LL_RTC_TIME_SetHour

Function name	**__STATIC_INLINE void LL_RTC_TIME_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)**
Function description	Set Hours in BCD format.
Parameters	• **RTCx:** RTC Instance • **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection

	function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function) • helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert hour from binary to BCD format
Reference Manual to LL API cross reference:	• TR HT LL_RTC_TIME_SetHour • TR HU LL_RTC_TIME_SetHour

LL_RTC_TIME_GetHour

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_GetHour (RTC_TypeDef * RTCx)**
Function description	Get Hours in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Notes	• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit • Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)). • helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert hour from BCD to Binary format
Reference Manual to LL API cross reference:	• TR HT LL_RTC_TIME_GetHour • TR HU LL_RTC_TIME_GetHour

LL_RTC_TIME_SetMinute

Function name	**__STATIC_INLINE void LL_RTC_TIME_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)**
Function description	Set Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance • **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function) • helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format
Reference Manual to LL API cross reference:	• TR MNT LL_RTC_TIME_SetMinute • TR MNU LL_RTC_TIME_SetMinute

LL_RTC_TIME_GetMinute

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_GetMinute (RTC_TypeDef * RTCx)**
Function description	Get Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit • Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)). • helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert minute from BCD to Binary format
Reference Manual to LL API cross reference:	• TR MNT LL_RTC_TIME_GetMinute • TR MNU LL_RTC_TIME_GetMinute

LL_RTC_TIME_SetSecond

Function name	**__STATIC_INLINE void LL_RTC_TIME_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)**
Function description	Set Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function) • helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format
Reference Manual to LL API cross reference:	• TR ST LL_RTC_TIME_SetSecond • TR SU LL_RTC_TIME_SetSecond

LL_RTC_TIME_GetSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_GetSecond (RTC_TypeDef * RTCx)**
Function description	Get Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit • Read either RTC_SSR or RTC_TR locks the values in the

	higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)). • helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format
Reference Manual to LL API cross reference:	• TR ST LL_RTC_TIME_GetSecond • TR SU LL_RTC_TIME_GetSecond

LL_RTC_TIME_Config

Function name	**__STATIC_INLINE void LL_RTC_TIME_Config (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)**
Function description	Set time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance • **Format12_24:** This parameter can be one of the following values: – LL_RTC_TIME_FORMAT_AM_OR_24 – LL_RTC_TIME_FORMAT_PM • **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23 • **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59 • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnableInitMode function) • TimeFormat and Hours should follow the same format
Reference Manual to LL API cross reference:	• TR PM LL_RTC_TIME_Config • TR HT LL_RTC_TIME_Config • TR HU LL_RTC_TIME_Config • TR MNT LL_RTC_TIME_Config • TR MNU LL_RTC_TIME_Config • TR ST LL_RTC_TIME_Config • TR SU LL_RTC_TIME_Config

LL_RTC_TIME_Get

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_Get (RTC_TypeDef * RTCx)**
Function description	Get time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Combination:** of hours, minutes and seconds (Format: 0x00HHMMSS).
Notes	• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit

	• Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)). • helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.
Reference Manual to LL API cross reference:	• TR HT LL_RTC_TIME_Get • TR HU LL_RTC_TIME_Get • TR MNT LL_RTC_TIME_Get • TR MNU LL_RTC_TIME_Get • TR ST LL_RTC_TIME_Get • TR SU LL_RTC_TIME_Get

LL_RTC_TIME_EnableDayLightStore

Function name	**__STATIC_INLINE void LL_RTC_TIME_EnableDayLightStore (RTC_TypeDef * RTCx)**
Function description	Memorize whether the daylight saving time change has been performed.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR BKP LL_RTC_TIME_EnableDayLightStore

LL_RTC_TIME_DisableDayLightStore

Function name	**__STATIC_INLINE void LL_RTC_TIME_DisableDayLightStore (RTC_TypeDef * RTCx)**
Function description	Disable memorization whether the daylight saving time change has been performed.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR BKP LL_RTC_TIME_DisableDayLightStore

LL_RTC_TIME_IsDayLightStoreEnabled

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_IsDayLightStoreEnabled (RTC_TypeDef * RTCx)**
Function description	Check if RTC Day Light Saving stored operation has been enabled

or not.

Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR BKP LL_RTC_TIME_IsDayLightStoreEnabled

LL_RTC_TIME_DecHour

Function name	**__STATIC_INLINE void LL_RTC_TIME_DecHour (RTC_TypeDef * RTCx)**
Function description	Subtract 1 hour (winter time change)
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR SUB1H LL_RTC_TIME_DecHour

LL_RTC_TIME_IncHour

Function name	**__STATIC_INLINE void LL_RTC_TIME_IncHour (RTC_TypeDef * RTCx)**
Function description	Add 1 hour (summer time change)
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ADD1H LL_RTC_TIME_IncHour

LL_RTC_TIME_GetSubSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_GetSubSecond (RTC_TypeDef * RTCx)**
Function description	Get Sub second value in the synchronous prescaler counter.
Parameters	• **RTCx:** RTC Instance
Return values	• **Sub:** second value (number between 0 and 65535)
Notes	• You can use both SubSeconds value and SecondFraction (PREDIV_S through LL_RTC_GetSynchPrescaler function) terms returned to convert Calendar SubSeconds value in second fraction ratio with time unit following generic formula: ==> Seconds fraction ratio * time_unit= [(SecondFraction-

		SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS.
Reference Manual to LL API cross reference:	•	SSR SS LL_RTC_TIME_GetSubSecond

LL_RTC_TIME_Synchronize

Function name	**__STATIC_INLINE void LL_RTC_TIME_Synchronize (RTC_TypeDef * RTCx, uint32_t ShiftSecond, uint32_t Fraction)**
Function description	Synchronize to a remote clock with a high degree of precision.
Parameters	• **RTCx:** RTC Instance • **ShiftSecond:** This parameter can be one of the following values: – LL_RTC_SHIFT_SECOND_DELAY – LL_RTC_SHIFT_SECOND_ADVANCE • **Fraction:** Number of Seconds Fractions (any value from 0 to 0x7FFF)
Return values	• **None:**
Notes	• This operation effectively subtracts from (delays) or advance the clock of a fraction of a second. • Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • When REFCKON is set, firmware must not write to Shift control register.
Reference Manual to LL API cross reference:	• SHIFTR ADD1S LL_RTC_TIME_Synchronize • SHIFTR SUBFS LL_RTC_TIME_Synchronize

LL_RTC_DATE_SetYear

Function name	**__STATIC_INLINE void LL_RTC_DATE_SetYear (RTC_TypeDef * RTCx, uint32_t Year)**
Function description	Set Year in BCD format.
Parameters	• **RTCx:** RTC Instance • **Year:** Value between Min_Data=0x00 and Max_Data=0x99
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Year from binary to BCD format
Reference Manual to LL API cross reference:	• DR YT LL_RTC_DATE_SetYear • DR YU LL_RTC_DATE_SetYear

LL_RTC_DATE_GetYear

Function name	**__STATIC_INLINE uint32_t LL_RTC_DATE_GetYear**

	(RTC_TypeDef * RTCx)
Function description	Get Year in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x99
Notes	• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit • helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Year from BCD to Binary format
Reference Manual to LL API cross reference:	• DR YT LL_RTC_DATE_GetYear • DR YU LL_RTC_DATE_GetYear

LL_RTC_DATE_SetWeekDay

Function name	**__STATIC_INLINE void LL_RTC_DATE_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)**
Function description	Set Week day.
Parameters	• **RTCx:** RTC Instance • **WeekDay:** This parameter can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR WDU LL_RTC_DATE_SetWeekDay

LL_RTC_DATE_GetWeekDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_DATE_GetWeekDay (RTC_TypeDef * RTCx)**
Function description	Get Week day.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY

Notes	• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
Reference Manual to LL API cross reference:	• DR WDU LL_RTC_DATE_GetWeekDay

LL_RTC_DATE_SetMonth

Function name	**__STATIC_INLINE void LL_RTC_DATE_SetMonth (RTC_TypeDef * RTCx, uint32_t Month)**
Function description	Set Month in BCD format.
Parameters	• **RTCx:** RTC Instance • **Month:** This parameter can be one of the following values: – LL_RTC_MONTH_JANUARY – LL_RTC_MONTH_FEBRUARY – LL_RTC_MONTH_MARCH – LL_RTC_MONTH_APRIL – LL_RTC_MONTH_MAY – LL_RTC_MONTH_JUNE – LL_RTC_MONTH_JULY – LL_RTC_MONTH_AUGUST – LL_RTC_MONTH_SEPTEMBER – LL_RTC_MONTH_OCTOBER – LL_RTC_MONTH_NOVEMBER – LL_RTC_MONTH_DECEMBER
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Month from binary to BCD format
Reference Manual to LL API cross reference:	• DR MT LL_RTC_DATE_SetMonth • DR MU LL_RTC_DATE_SetMonth

LL_RTC_DATE_GetMonth

Function name	**__STATIC_INLINE uint32_t LL_RTC_DATE_GetMonth (RTC_TypeDef * RTCx)**
Function description	Get Month in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_MONTH_JANUARY – LL_RTC_MONTH_FEBRUARY – LL_RTC_MONTH_MARCH – LL_RTC_MONTH_APRIL – LL_RTC_MONTH_MAY – LL_RTC_MONTH_JUNE – LL_RTC_MONTH_JULY – LL_RTC_MONTH_AUGUST – LL_RTC_MONTH_SEPTEMBER – LL_RTC_MONTH_OCTOBER

	– LL_RTC_MONTH_NOVEMBER – LL_RTC_MONTH_DECEMBER
Notes	• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit • helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format
Reference Manual to LL API cross reference:	• DR MT LL_RTC_DATE_GetMonth • DR MU LL_RTC_DATE_GetMonth

LL_RTC_DATE_SetDay

Function name	**__STATIC_INLINE void LL_RTC_DATE_SetDay (RTC_TypeDef * RTCx, uint32_t Day)**
Function description	Set Day in BCD format.
Parameters	• **RTCx:** RTC Instance • **Day:** Value between Min_Data=0x01 and Max_Data=0x31
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format
Reference Manual to LL API cross reference:	• DR DT LL_RTC_DATE_SetDay • DR DU LL_RTC_DATE_SetDay

LL_RTC_DATE_GetDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_DATE_GetDay (RTC_TypeDef * RTCx)**
Function description	Get Day in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x31
Notes	• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit • helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format
Reference Manual to LL API cross reference:	• DR DT LL_RTC_DATE_GetDay • DR DU LL_RTC_DATE_GetDay

LL_RTC_DATE_Config

Function name	**__STATIC_INLINE void LL_RTC_DATE_Config (RTC_TypeDef * RTCx, uint32_t WeekDay, uint32_t Day, uint32_t Month, uint32_t Year)**
Function description	Set date (WeekDay, Day, Month and Year) in BCD format.
Parameters	• **RTCx:** RTC Instance • **WeekDay:** This parameter can be one of the following

values:
- LL_RTC_WEEKDAY_MONDAY
- LL_RTC_WEEKDAY_TUESDAY
- LL_RTC_WEEKDAY_WEDNESDAY
- LL_RTC_WEEKDAY_THURSDAY
- LL_RTC_WEEKDAY_FRIDAY
- LL_RTC_WEEKDAY_SATURDAY
- LL_RTC_WEEKDAY_SUNDAY

- **Day:** Value between Min_Data=0x01 and Max_Data=0x31
- **Month:** This parameter can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values
- **None:**

Reference Manual to LL API cross reference:
- DR WDU LL_RTC_DATE_Config
- DR MT LL_RTC_DATE_Config
- DR MU LL_RTC_DATE_Config
- DR DT LL_RTC_DATE_Config
- DR DU LL_RTC_DATE_Config
- DR YT LL_RTC_DATE_Config
- DR YU LL_RTC_DATE_Config

LL_RTC_DATE_Get

Function name: **__STATIC_INLINE uint32_t LL_RTC_DATE_Get (RTC_TypeDef * RTCx)**

Function description: Get date (WeekDay, Day, Month and Year) in BCD format.

Parameters
- **RTCx:** RTC Instance

Return values
- **Combination:** of WeekDay, Day, Month and Year (Format: 0xWWDDMMYY).

Notes
- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_YEAR, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:
- DR WDU LL_RTC_DATE_Get
- DR MT LL_RTC_DATE_Get
- DR MU LL_RTC_DATE_Get

- DR DT LL_RTC_DATE_Get
- DR DU LL_RTC_DATE_Get
- DR YT LL_RTC_DATE_Get
- DR YU LL_RTC_DATE_Get

LL_RTC_ALMA_Enable

Function name	**__STATIC_INLINE void LL_RTC_ALMA_Enable (RTC_TypeDef * RTCx)**
Function description	Enable Alarm A.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRAE LL_RTC_ALMA_Enable

LL_RTC_ALMA_Disable

Function name	**__STATIC_INLINE void LL_RTC_ALMA_Disable (RTC_TypeDef * RTCx)**
Function description	Disable Alarm A.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRAE LL_RTC_ALMA_Disable

LL_RTC_ALMA_SetMask

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Specify the Alarm A masks.
Parameters	• **RTCx:** RTC Instance • **Mask:** This parameter can be a combination of the following values: – LL_RTC_ALMA_MASK_NONE – LL_RTC_ALMA_MASK_DATEWEEKDAY – LL_RTC_ALMA_MASK_HOURS – LL_RTC_ALMA_MASK_MINUTES – LL_RTC_ALMA_MASK_SECONDS – LL_RTC_ALMA_MASK_ALL
Return values	• **None:**

Reference Manual to LL API cross reference:	• ALRMAR MSK4 LL_RTC_ALMA_SetMask • ALRMAR MSK3 LL_RTC_ALMA_SetMask • ALRMAR MSK2 LL_RTC_ALMA_SetMask • ALRMAR MSK1 LL_RTC_ALMA_SetMask

LL_RTC_ALMA_GetMask

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMask (RTC_TypeDef * RTCx)**
Function description	Get the Alarm A masks.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be can be a combination of the following values: – LL_RTC_ALMA_MASK_NONE – LL_RTC_ALMA_MASK_DATEWEEKDAY – LL_RTC_ALMA_MASK_HOURS – LL_RTC_ALMA_MASK_MINUTES – LL_RTC_ALMA_MASK_SECONDS – LL_RTC_ALMA_MASK_ALL
Reference Manual to LL API cross reference:	• ALRMAR MSK4 LL_RTC_ALMA_GetMask • ALRMAR MSK3 LL_RTC_ALMA_GetMask • ALRMAR MSK2 LL_RTC_ALMA_GetMask • ALRMAR MSK1 LL_RTC_ALMA_GetMask

LL_RTC_ALMA_EnableWeekday

Function name	**__STATIC_INLINE void LL_RTC_ALMA_EnableWeekday (RTC_TypeDef * RTCx)**
Function description	Enable AlarmA Week day selection (DU[3:0] represents the week day.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMAR WDSEL LL_RTC_ALMA_EnableWeekday

LL_RTC_ALMA_DisableWeekday

Function name	**__STATIC_INLINE void LL_RTC_ALMA_DisableWeekday (RTC_TypeDef * RTCx)**
Function description	Disable AlarmA Week day selection (DU[3:0] represents the date)
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMAR WDSEL LL_RTC_ALMA_DisableWeekday

LL_RTC_ALMA_SetDay

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetDay (RTC_TypeDef * RTCx, uint32_t Day)**
Function description	Set ALARM A Day in BCD format.
Parameters	• **RTCx:** RTC Instance • **Day:** Value between Min_Data=0x01 and Max_Data=0x31
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMAR DT LL_RTC_ALMA_SetDay • ALRMAR DU LL_RTC_ALMA_SetDay

LL_RTC_ALMA_GetDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetDay (RTC_TypeDef * RTCx)**
Function description	Get ALARM A Day in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x31
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMAR DT LL_RTC_ALMA_GetDay • ALRMAR DU LL_RTC_ALMA_GetDay

LL_RTC_ALMA_SetWeekDay

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)**
Function description	Set ALARM A Weekday.
Parameters	• **RTCx:** RTC Instance • **WeekDay:** This parameter can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMAR DU LL_RTC_ALMA_SetWeekDay

LL_RTC_ALMA_GetWeekDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetWeekDay (RTC_TypeDef * RTCx)**
Function description	Get ALARM A Weekday.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY
Reference Manual to LL API cross reference:	• ALRMAR DU LL_RTC_ALMA_GetWeekDay

LL_RTC_ALMA_SetTimeFormat

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)**
Function description	Set Alarm A time format (AM/24-hour or PM notation)
Parameters	• **RTCx:** RTC Instance • **TimeFormat:** This parameter can be one of the following values: – LL_RTC_ALMA_TIME_FORMAT_AM – LL_RTC_ALMA_TIME_FORMAT_PM
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMAR PM LL_RTC_ALMA_SetTimeFormat

LL_RTC_ALMA_GetTimeFormat

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTimeFormat (RTC_TypeDef * RTCx)**
Function description	Get Alarm A time format (AM or PM notation)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_ALMA_TIME_FORMAT_AM – LL_RTC_ALMA_TIME_FORMAT_PM
Reference Manual to LL API cross reference:	• ALRMAR PM LL_RTC_ALMA_GetTimeFormat

LL_RTC_ALMA_SetHour

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)**
Function description	Set ALARM A Hours in BCD format.
Parameters	• **RTCx:** RTC Instance • **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMAR HT LL_RTC_ALMA_SetHour • ALRMAR HU LL_RTC_ALMA_SetHour

LL_RTC_ALMA_GetHour

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetHour (RTC_TypeDef * RTCx)**
Function description	Get ALARM A Hours in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMAR HT LL_RTC_ALMA_GetHour • ALRMAR HU LL_RTC_ALMA_GetHour

LL_RTC_ALMA_SetMinute

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)**
Function description	Set ALARM A Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance • **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMAR MNT LL_RTC_ALMA_SetMinute • ALRMAR MNU LL_RTC_ALMA_SetMinute

LL_RTC_ALMA_GetMinute

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMinute (RTC_TypeDef * RTCx)**
Function description	Get ALARM A Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMAR MNT LL_RTC_ALMA_GetMinute • ALRMAR MNU LL_RTC_ALMA_GetMinute

LL_RTC_ALMA_SetSecond

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)**
Function description	Set ALARM A Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMAR ST LL_RTC_ALMA_SetSecond • ALRMAR SU LL_RTC_ALMA_SetSecond

LL_RTC_ALMA_GetSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSecond (RTC_TypeDef * RTCx)**
Function description	Get ALARM A Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMAR ST LL_RTC_ALMA_GetSecond • ALRMAR SU LL_RTC_ALMA_GetSecond

LL_RTC_ALMA_ConfigTime

Function name	**__STATIC_INLINE void LL_RTC_ALMA_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)**

Function description	Set Alarm A Time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance • **Format12_24:** This parameter can be one of the following values: – LL_RTC_ALMA_TIME_FORMAT_AM – LL_RTC_ALMA_TIME_FORMAT_PM • **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23 • **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59 • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMAR PM LL_RTC_ALMA_ConfigTime • ALRMAR HT LL_RTC_ALMA_ConfigTime • ALRMAR HU LL_RTC_ALMA_ConfigTime • ALRMAR MNT LL_RTC_ALMA_ConfigTime • ALRMAR MNU LL_RTC_ALMA_ConfigTime • ALRMAR ST LL_RTC_ALMA_ConfigTime • ALRMAR SU LL_RTC_ALMA_ConfigTime

LL_RTC_ALMA_GetTime

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTime (RTC_TypeDef * RTCx)**
Function description	Get Alarm B Time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Combination:** of hours, minutes and seconds.
Notes	• helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.
Reference Manual to LL API cross reference:	• ALRMAR HT LL_RTC_ALMA_GetTime • ALRMAR HU LL_RTC_ALMA_GetTime • ALRMAR MNT LL_RTC_ALMA_GetTime • ALRMAR MNU LL_RTC_ALMA_GetTime • ALRMAR ST LL_RTC_ALMA_GetTime • ALRMAR SU LL_RTC_ALMA_GetTime

LL_RTC_ALMA_SetSubSecondMask

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Set Alarm A Mask the most-significant bits starting at this bit.
Parameters	• **RTCx:** RTC Instance • **Mask:** Value between Min_Data=0x00 and Max_Data=0xF
Return values	• **None:**
Notes	• This register can be written only when ALRAE is reset in

	RTC_CR register, or in initialization mode.
Reference Manual to LL API cross reference:	• ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask

LL_RTC_ALMA_GetSubSecondMask

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecondMask (RTC_TypeDef * RTCx)**
Function description	Get Alarm A Mask the most-significant bits starting at this bit.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xF
Reference Manual to LL API cross reference:	• ALRMASSR MASKSS LL_RTC_ALMA_GetSubSecondMask

LL_RTC_ALMA_SetSubSecond

Function name	**__STATIC_INLINE void LL_RTC_ALMA_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)**
Function description	Set Alarm A Sub seconds value.
Parameters	• **RTCx:** RTC Instance • **Subsecond:** Value between Min_Data=0x00 and Max_Data=0x7FFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMASSR SS LL_RTC_ALMA_SetSubSecond

LL_RTC_ALMA_GetSubSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond (RTC_TypeDef * RTCx)**
Function description	Get Alarm A Sub seconds value.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x7FFF
Reference Manual to LL API cross reference:	• ALRMASSR SS LL_RTC_ALMA_GetSubSecond

LL_RTC_ALMB_Enable

Function name	**__STATIC_INLINE void LL_RTC_ALMB_Enable (RTC_TypeDef * RTCx)**
Function description	Enable Alarm B.
Parameters	• **RTCx:** RTC Instance

Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRBE LL_RTC_ALMB_Enable

LL_RTC_ALMB_Disable

Function name	**__STATIC_INLINE void LL_RTC_ALMB_Disable (RTC_TypeDef * RTCx)**
Function description	Disable Alarm B.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRBE LL_RTC_ALMB_Disable

LL_RTC_ALMB_SetMask

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Specify the Alarm B masks.
Parameters	• **RTCx:** RTC Instance • **Mask:** This parameter can be a combination of the following values: – LL_RTC_ALMB_MASK_NONE – LL_RTC_ALMB_MASK_DATEWEEKDAY – LL_RTC_ALMB_MASK_HOURS – LL_RTC_ALMB_MASK_MINUTES – LL_RTC_ALMB_MASK_SECONDS – LL_RTC_ALMB_MASK_ALL
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMBR MSK4 LL_RTC_ALMB_SetMask • ALRMBR MSK3 LL_RTC_ALMB_SetMask • ALRMBR MSK2 LL_RTC_ALMB_SetMask • ALRMBR MSK1 LL_RTC_ALMB_SetMask

LL_RTC_ALMB_GetMask

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMask (RTC_TypeDef * RTCx)**
Function description	Get the Alarm B masks.
Parameters	• **RTCx:** RTC Instance

Return values	• **Returned:** value can be can be a combination of the following values: – LL_RTC_ALMB_MASK_NONE – LL_RTC_ALMB_MASK_DATEWEEKDAY – LL_RTC_ALMB_MASK_HOURS – LL_RTC_ALMB_MASK_MINUTES – LL_RTC_ALMB_MASK_SECONDS – LL_RTC_ALMB_MASK_ALL
Reference Manual to LL API cross reference:	• ALRMBR MSK4 LL_RTC_ALMB_GetMask • ALRMBR MSK3 LL_RTC_ALMB_GetMask • ALRMBR MSK2 LL_RTC_ALMB_GetMask • ALRMBR MSK1 LL_RTC_ALMB_GetMask

LL_RTC_ALMB_EnableWeekday

Function name	**__STATIC_INLINE void LL_RTC_ALMB_EnableWeekday (RTC_TypeDef * RTCx)**
Function description	Enable AlarmB Week day selection (DU[3:0] represents the week day.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMBR WDSEL LL_RTC_ALMB_EnableWeekday

LL_RTC_ALMB_DisableWeekday

Function name	**__STATIC_INLINE void LL_RTC_ALMB_DisableWeekday (RTC_TypeDef * RTCx)**
Function description	Disable AlarmB Week day selection (DU[3:0] represents the date)
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMBR WDSEL LL_RTC_ALMB_DisableWeekday

LL_RTC_ALMB_SetDay

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetDay (RTC_TypeDef * RTCx, uint32_t Day)**
Function description	Set ALARM B Day in BCD format.
Parameters	• **RTCx:** RTC Instance • **Day:** Value between Min_Data=0x01 and Max_Data=0x31
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:	• ALRMBR DT LL_RTC_ALMB_SetDay • ALRMBR DU LL_RTC_ALMB_SetDay

LL_RTC_ALMB_GetDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetDay (RTC_TypeDef * RTCx)**
Function description	Get ALARM B Day in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x31
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMBR DT LL_RTC_ALMB_GetDay • ALRMBR DU LL_RTC_ALMB_GetDay

LL_RTC_ALMB_SetWeekDay

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)**
Function description	Set ALARM B Weekday.
Parameters	• **RTCx:** RTC Instance • **WeekDay:** This parameter can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMBR DU LL_RTC_ALMB_SetWeekDay

LL_RTC_ALMB_GetWeekDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetWeekDay (RTC_TypeDef * RTCx)**
Function description	Get ALARM B Weekday.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY

– LL_RTC_WEEKDAY_FRIDAY
– LL_RTC_WEEKDAY_SATURDAY
– LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:	• ALRMBR DU LL_RTC_ALMB_GetWeekDay

LL_RTC_ALMB_SetTimeFormat

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)**
Function description	Set ALARM B time format (AM/24-hour or PM notation)
Parameters	• **RTCx:** RTC Instance • **TimeFormat:** This parameter can be one of the following values: – LL_RTC_ALMB_TIME_FORMAT_AM – LL_RTC_ALMB_TIME_FORMAT_PM
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMBR PM LL_RTC_ALMB_SetTimeFormat

LL_RTC_ALMB_GetTimeFormat

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTimeFormat (RTC_TypeDef * RTCx)**
Function description	Get ALARM B time format (AM or PM notation)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_ALMB_TIME_FORMAT_AM – LL_RTC_ALMB_TIME_FORMAT_PM
Reference Manual to LL API cross reference:	• ALRMBR PM LL_RTC_ALMB_GetTimeFormat

LL_RTC_ALMB_SetHour

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)**
Function description	Set ALARM B Hours in BCD format.
Parameters	• **RTCx:** RTC Instance • **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format
Reference Manual to	• ALRMBR HT LL_RTC_ALMB_SetHour

LL API cross reference:	• ALRMBR HU LL_RTC_ALMB_SetHour

LL_RTC_ALMB_GetHour

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetHour (RTC_TypeDef * RTCx)**
Function description	Get ALARM B Hours in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMBR HT LL_RTC_ALMB_GetHour • ALRMBR HU LL_RTC_ALMB_GetHour

LL_RTC_ALMB_SetMinute

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)**
Function description	Set ALARM B Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance • **Minutes:** between Min_Data=0x00 and Max_Data=0x59
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMBR MNT LL_RTC_ALMB_SetMinute • ALRMBR MNU LL_RTC_ALMB_SetMinute

LL_RTC_ALMB_GetMinute

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMinute (RTC_TypeDef * RTCx)**
Function description	Get ALARM B Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMBR MNT LL_RTC_ALMB_GetMinute • ALRMBR MNU LL_RTC_ALMB_GetMinute

LL_RTC_ALMB_SetSecond

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)**
Function description	Set ALARM B Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None:**
Notes	• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format
Reference Manual to LL API cross reference:	• ALRMBR ST LL_RTC_ALMB_SetSecond • ALRMBR SU LL_RTC_ALMB_SetSecond

LL_RTC_ALMB_GetSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSecond (RTC_TypeDef * RTCx)**
Function description	Get ALARM B Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format
Reference Manual to LL API cross reference:	• ALRMBR ST LL_RTC_ALMB_GetSecond • ALRMBR SU LL_RTC_ALMB_GetSecond

LL_RTC_ALMB_ConfigTime

Function name	**__STATIC_INLINE void LL_RTC_ALMB_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)**
Function description	Set Alarm B Time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance • **Format12_24:** This parameter can be one of the following values: – LL_RTC_ALMB_TIME_FORMAT_AM – LL_RTC_ALMB_TIME_FORMAT_PM • **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23 • **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59 • **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59
Return values	• **None:**
Reference Manual to	• ALRMBR PM LL_RTC_ALMB_ConfigTime

LL API cross reference:	• ALRMBR HT LL_RTC_ALMB_ConfigTime • ALRMBR HU LL_RTC_ALMB_ConfigTime • ALRMBR MNT LL_RTC_ALMB_ConfigTime • ALRMBR MNU LL_RTC_ALMB_ConfigTime • ALRMBR ST LL_RTC_ALMB_ConfigTime • ALRMBR SU LL_RTC_ALMB_ConfigTime

LL_RTC_ALMB_GetTime

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTime (RTC_TypeDef * RTCx)**
Function description	Get Alarm B Time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Combination:** of hours, minutes and seconds.
Notes	• helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.
Reference Manual to LL API cross reference:	• ALRMBR HT LL_RTC_ALMB_GetTime • ALRMBR HU LL_RTC_ALMB_GetTime • ALRMBR MNT LL_RTC_ALMB_GetTime • ALRMBR MNU LL_RTC_ALMB_GetTime • ALRMBR ST LL_RTC_ALMB_GetTime • ALRMBR SU LL_RTC_ALMB_GetTime

LL_RTC_ALMB_SetSubSecondMask

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Set Alarm B Mask the most-significant bits starting at this bit.
Parameters	• **RTCx:** RTC Instance • **Mask:** Value between Min_Data=0x00 and Max_Data=0xF
Return values	• **None:**
Notes	• This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.
Reference Manual to LL API cross reference:	• ALRMBSSR MASKSS LL_RTC_ALMB_SetSubSecondMask

LL_RTC_ALMB_GetSubSecondMask

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecondMask (RTC_TypeDef * RTCx)**
Function description	Get Alarm B Mask the most-significant bits starting at this bit.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xF
Reference Manual to	• ALRMBSSR MASKSS LL_RTC_ALMB_GetSubSecondMask

LL API cross reference:

LL_RTC_ALMB_SetSubSecond

Function name	**__STATIC_INLINE void LL_RTC_ALMB_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)**
Function description	Set Alarm B Sub seconds value.
Parameters	• **RTCx:** RTC Instance • **Subsecond:** Value between Min_Data=0x00 and Max_Data=0x7FFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• ALRMBSSR SS LL_RTC_ALMB_SetSubSecond

LL_RTC_ALMB_GetSubSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecond (RTC_TypeDef * RTCx)**
Function description	Get Alarm B Sub seconds value.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x7FFF
Reference Manual to LL API cross reference:	• ALRMBSSR SS LL_RTC_ALMB_GetSubSecond

LL_RTC_TS_EnableInternalEvent

Function name	**__STATIC_INLINE void LL_RTC_TS_EnableInternalEvent (RTC_TypeDef * RTCx)**
Function description	Enable internal event timestamp.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ITSE LL_RTC_TS_EnableInternalEvent

LL_RTC_TS_DisableInternalEvent

Function name	**__STATIC_INLINE void LL_RTC_TS_DisableInternalEvent (RTC_TypeDef * RTCx)**
Function description	Disable internal event timestamp.
Parameters	• **RTCx:** RTC Instance

Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ITSE LL_RTC_TS_DisableInternalEvent

LL_RTC_TS_Enable

Function name	**__STATIC_INLINE void LL_RTC_TS_Enable (RTC_TypeDef * RTCx)**
Function description	Enable Timestamp.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR TSE LL_RTC_TS_Enable

LL_RTC_TS_Disable

Function name	**__STATIC_INLINE void LL_RTC_TS_Disable (RTC_TypeDef * RTCx)**
Function description	Disable Timestamp.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR TSE LL_RTC_TS_Disable

LL_RTC_TS_SetActiveEdge

Function name	**__STATIC_INLINE void LL_RTC_TS_SetActiveEdge (RTC_TypeDef * RTCx, uint32_t Edge)**
Function description	Set Time-stamp event active edge.
Parameters	• **RTCx:** RTC Instance • **Edge:** This parameter can be one of the following values: – LL_RTC_TIMESTAMP_EDGE_RISING – LL_RTC_TIMESTAMP_EDGE_FALLING
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • TSE must be reset when TSEDGE is changed to avoid

	unwanted TSF setting
Reference Manual to LL API cross reference:	• CR TSEDGE LL_RTC_TS_SetActiveEdge

LL_RTC_TS_GetActiveEdge

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetActiveEdge (RTC_TypeDef * RTCx)**
Function description	Get Time-stamp event active edge.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_TIMESTAMP_EDGE_RISING – LL_RTC_TIMESTAMP_EDGE_FALLING
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR TSEDGE LL_RTC_TS_GetActiveEdge

LL_RTC_TS_GetTimeFormat

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetTimeFormat (RTC_TypeDef * RTCx)**
Function description	Get Timestamp AM/PM notation (AM or 24-hour format)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_TS_TIME_FORMAT_AM – LL_RTC_TS_TIME_FORMAT_PM
Reference Manual to LL API cross reference:	• TSTR PM LL_RTC_TS_GetTimeFormat

LL_RTC_TS_GetHour

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetHour (RTC_TypeDef * RTCx)**
Function description	Get Timestamp Hours in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format
Reference Manual to LL API cross reference:	• TSTR HT LL_RTC_TS_GetHour • TSTR HU LL_RTC_TS_GetHour

LL_RTC_TS_GetMinute

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetMinute (RTC_TypeDef * RTCx)**
Function description	Get Timestamp Minutes in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format
Reference Manual to LL API cross reference:	• TSTR MNT LL_RTC_TS_GetMinute • TSTR MNU LL_RTC_TS_GetMinute

LL_RTC_TS_GetSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetSecond (RTC_TypeDef * RTCx)**
Function description	Get Timestamp Seconds in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x59
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format
Reference Manual to LL API cross reference:	• TSTR ST LL_RTC_TS_GetSecond • TSTR SU LL_RTC_TS_GetSecond

LL_RTC_TS_GetTime

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetTime (RTC_TypeDef * RTCx)**
Function description	Get Timestamp time (hour, minute and second) in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Combination:** of hours, minutes and seconds.
Notes	• helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.
Reference Manual to LL API cross reference:	• TSTR HT LL_RTC_TS_GetTime • TSTR HU LL_RTC_TS_GetTime • TSTR MNT LL_RTC_TS_GetTime • TSTR MNU LL_RTC_TS_GetTime • TSTR ST LL_RTC_TS_GetTime • TSTR SU LL_RTC_TS_GetTime

LL_RTC_TS_GetWeekDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetWeekDay**

	(RTC_TypeDef * RTCx)
Function description	Get Timestamp Week day.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_WEEKDAY_MONDAY – LL_RTC_WEEKDAY_TUESDAY – LL_RTC_WEEKDAY_WEDNESDAY – LL_RTC_WEEKDAY_THURSDAY – LL_RTC_WEEKDAY_FRIDAY – LL_RTC_WEEKDAY_SATURDAY – LL_RTC_WEEKDAY_SUNDAY
Reference Manual to LL API cross reference:	• TSDR WDU LL_RTC_TS_GetWeekDay

LL_RTC_TS_GetMonth

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetMonth (RTC_TypeDef * RTCx)**
Function description	Get Timestamp Month in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_MONTH_JANUARY – LL_RTC_MONTH_FEBRUARY – LL_RTC_MONTH_MARCH – LL_RTC_MONTH_APRIL – LL_RTC_MONTH_MAY – LL_RTC_MONTH_JUNE – LL_RTC_MONTH_JULY – LL_RTC_MONTH_AUGUST – LL_RTC_MONTH_SEPTEMBER – LL_RTC_MONTH_OCTOBER – LL_RTC_MONTH_NOVEMBER – LL_RTC_MONTH_DECEMBER
Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format
Reference Manual to LL API cross reference:	• TSDR MT LL_RTC_TS_GetMonth • TSDR MU LL_RTC_TS_GetMonth

LL_RTC_TS_GetDay

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetDay (RTC_TypeDef * RTCx)**
Function description	Get Timestamp Day in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes	• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format
Reference Manual to LL API cross reference:	• TSDR DT LL_RTC_TS_GetDay • TSDR DU LL_RTC_TS_GetDay

LL_RTC_TS_GetDate

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetDate (RTC_TypeDef * RTCx)**
Function description	Get Timestamp date (WeekDay, Day and Month) in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Combination:** of Weekday, Day and Month
Notes	• helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.
Reference Manual to LL API cross reference:	• TSDR WDU LL_RTC_TS_GetDate • TSDR MT LL_RTC_TS_GetDate • TSDR MU LL_RTC_TS_GetDate • TSDR DT LL_RTC_TS_GetDate • TSDR DU LL_RTC_TS_GetDate

LL_RTC_TS_GetSubSecond

Function name	**__STATIC_INLINE uint32_t LL_RTC_TS_GetSubSecond (RTC_TypeDef * RTCx)**
Function description	Get time-stamp sub second value.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• TSSSR SS LL_RTC_TS_GetSubSecond

LL_RTC_TS_EnableOnTamper

Function name	**__STATIC_INLINE void LL_RTC_TS_EnableOnTamper (RTC_TypeDef * RTCx)**
Function description	Activate timestamp on tamper detection event.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMPTS LL_RTC_TS_EnableOnTamper

LL_RTC_TS_DisableOnTamper

Function name	**__STATIC_INLINE void LL_RTC_TS_DisableOnTamper (RTC_TypeDef * RTCx)**
Function description	Disable timestamp on tamper detection event.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMPTS LL_RTC_TS_DisableOnTamper

LL_RTC_TAMPER_Enable

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_Enable (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Enable RTC_TAMPx input detection.
Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: – LL_RTC_TAMPER_1 – LL_RTC_TAMPER_2 – LL_RTC_TAMPER_3
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1E LL_RTC_TAMPER_Enable • TAMPCR TAMP2E LL_RTC_TAMPER_Enable • TAMPCR TAMP3E LL_RTC_TAMPER_Enable

LL_RTC_TAMPER_Disable

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_Disable (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Clear RTC_TAMPx input detection.
Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: – LL_RTC_TAMPER_1 – LL_RTC_TAMPER_2 – LL_RTC_TAMPER_3
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1E LL_RTC_TAMPER_Disable • TAMPCR TAMP2E LL_RTC_TAMPER_Disable • TAMPCR TAMP3E LL_RTC_TAMPER_Disable

LL_RTC_TAMPER_EnableMask

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_EnableMask (RTC_TypeDef * RTCx, uint32_t Mask)**

Function description	Enable Tamper mask flag.
Parameters	• **RTCx:** RTC Instance • **Mask:** This parameter can be a combination of the following values: – LL_RTC_TAMPER_MASK_TAMPER1 – LL_RTC_TAMPER_MASK_TAMPER2 – LL_RTC_TAMPER_MASK_TAMPER3
Return values	• **None:**
Notes	• Associated Tamper IT must not enabled when tamper mask is set.
Reference Manual to LL API cross reference:	• TAMPCR TAMP1MF LL_RTC_TAMPER_EnableMask • TAMPCR TAMP2MF LL_RTC_TAMPER_EnableMask • TAMPCR TAMP3MF LL_RTC_TAMPER_EnableMask

LL_RTC_TAMPER_DisableMask

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_DisableMask (RTC_TypeDef * RTCx, uint32_t Mask)**
Function description	Disable Tamper mask flag.
Parameters	• **RTCx:** RTC Instance • **Mask:** This parameter can be a combination of the following values: – LL_RTC_TAMPER_MASK_TAMPER1 – LL_RTC_TAMPER_MASK_TAMPER2 – LL_RTC_TAMPER_MASK_TAMPER3
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1MF LL_RTC_TAMPER_DisableMask • TAMPCR TAMP2MF LL_RTC_TAMPER_DisableMask • TAMPCR TAMP3MF LL_RTC_TAMPER_DisableMask

LL_RTC_TAMPER_EnableEraseBKP

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_EnableEraseBKP (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Enable backup register erase after Tamper event detection.
Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: – LL_RTC_TAMPER_NOERASE_TAMPER1 – LL_RTC_TAMPER_NOERASE_TAMPER2 – LL_RTC_TAMPER_NOERASE_TAMPER3
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1NOERASE LL_RTC_TAMPER_EnableEraseBKP • TAMPCR TAMP2NOERASE LL_RTC_TAMPER_EnableEraseBKP • TAMPCR TAMP3NOERASE

LL_RTC_TAMPER_EnableEraseBKP

LL_RTC_TAMPER_DisableEraseBKP

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_DisableEraseBKP (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Disable backup register erase after Tamper event detection.
Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: – LL_RTC_TAMPER_NOERASE_TAMPER1 – LL_RTC_TAMPER_NOERASE_TAMPER2 – LL_RTC_TAMPER_NOERASE_TAMPER3
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1NOERASE LL_RTC_TAMPER_DisableEraseBKP • TAMPCR TAMP2NOERASE LL_RTC_TAMPER_DisableEraseBKP • TAMPCR TAMP3NOERASE LL_RTC_TAMPER_DisableEraseBKP

LL_RTC_TAMPER_DisablePullUp

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_DisablePullUp (RTC_TypeDef * RTCx)**
Function description	Disable RTC_TAMPx pull-up disable (Disable precharge of RTC_TAMPx pins)
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMPPUDIS LL_RTC_TAMPER_DisablePullUp

LL_RTC_TAMPER_EnablePullUp

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_EnablePullUp (RTC_TypeDef * RTCx)**
Function description	Enable RTC_TAMPx pull-up disable (Precharge RTC_TAMPx pins before sampling)
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMPPUDIS LL_RTC_TAMPER_EnablePullUp

LL_RTC_TAMPER_SetPrecharge

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_SetPrecharge (RTC_TypeDef * RTCx, uint32_t Duration)**
Function description	Set RTC_TAMPx precharge duration.
Parameters	• **RTCx:** RTC Instance • **Duration:** This parameter can be one of the following values: – LL_RTC_TAMPER_DURATION_1RTCCLK – LL_RTC_TAMPER_DURATION_2RTCCLK – LL_RTC_TAMPER_DURATION_4RTCCLK – LL_RTC_TAMPER_DURATION_8RTCCLK
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMPPRCH LL_RTC_TAMPER_SetPrecharge

LL_RTC_TAMPER_GetPrecharge

Function name	**__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetPrecharge (RTC_TypeDef * RTCx)**
Function description	Get RTC_TAMPx precharge duration.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_TAMPER_DURATION_1RTCCLK – LL_RTC_TAMPER_DURATION_2RTCCLK – LL_RTC_TAMPER_DURATION_4RTCCLK – LL_RTC_TAMPER_DURATION_8RTCCLK
Reference Manual to LL API cross reference:	• TAMPCR TAMPPRCH LL_RTC_TAMPER_GetPrecharge

LL_RTC_TAMPER_SetFilterCount

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_SetFilterCount (RTC_TypeDef * RTCx, uint32_t FilterCount)**
Function description	Set RTC_TAMPx filter count.
Parameters	• **RTCx:** RTC Instance • **FilterCount:** This parameter can be one of the following values: – LL_RTC_TAMPER_FILTER_DISABLE – LL_RTC_TAMPER_FILTER_2SAMPLE – LL_RTC_TAMPER_FILTER_4SAMPLE – LL_RTC_TAMPER_FILTER_8SAMPLE
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMPFLT LL_RTC_TAMPER_SetFilterCount

LL_RTC_TAMPER_GetFilterCount

Function name	**__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetFilterCount (RTC_TypeDef * RTCx)**
Function description	Get RTC_TAMPx filter count.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_TAMPER_FILTER_DISABLE – LL_RTC_TAMPER_FILTER_2SAMPLE – LL_RTC_TAMPER_FILTER_4SAMPLE – LL_RTC_TAMPER_FILTER_8SAMPLE
Reference Manual to LL API cross reference:	• TAMPCR TAMPFLT LL_RTC_TAMPER_GetFilterCount

LL_RTC_TAMPER_SetSamplingFreq

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_SetSamplingFreq (RTC_TypeDef * RTCx, uint32_t SamplingFreq)**
Function description	Set Tamper sampling frequency.
Parameters	• **RTCx:** RTC Instance • **SamplingFreq:** This parameter can be one of the following values: – LL_RTC_TAMPER_SAMPLFREQDIV_32768 – LL_RTC_TAMPER_SAMPLFREQDIV_16384 – LL_RTC_TAMPER_SAMPLFREQDIV_8192 – LL_RTC_TAMPER_SAMPLFREQDIV_4096 – LL_RTC_TAMPER_SAMPLFREQDIV_2048 – LL_RTC_TAMPER_SAMPLFREQDIV_1024 – LL_RTC_TAMPER_SAMPLFREQDIV_512 – LL_RTC_TAMPER_SAMPLFREQDIV_256
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMPFREQ LL_RTC_TAMPER_SetSamplingFreq

LL_RTC_TAMPER_GetSamplingFreq

Function name	**__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetSamplingFreq (RTC_TypeDef * RTCx)**
Function description	Get Tamper sampling frequency.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_TAMPER_SAMPLFREQDIV_32768 – LL_RTC_TAMPER_SAMPLFREQDIV_16384 – LL_RTC_TAMPER_SAMPLFREQDIV_8192 – LL_RTC_TAMPER_SAMPLFREQDIV_4096 – LL_RTC_TAMPER_SAMPLFREQDIV_2048

– LL_RTC_TAMPER_SAMPLFREQDIV_1024
– LL_RTC_TAMPER_SAMPLFREQDIV_512
– LL_RTC_TAMPER_SAMPLFREQDIV_256

Reference Manual to LL API cross reference:	• TAMPCR TAMPFREQ LL_RTC_TAMPER_GetSamplingFreq

LL_RTC_TAMPER_EnableActiveLevel

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_EnableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Enable Active level for Tamper input.
Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: – LL_RTC_TAMPER_ACTIVELEVEL_TAMP1 – LL_RTC_TAMPER_ACTIVELEVEL_TAMP2 – LL_RTC_TAMPER_ACTIVELEVEL_TAMP3
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1TRG LL_RTC_TAMPER_EnableActiveLevel • TAMPCR TAMP2TRG LL_RTC_TAMPER_EnableActiveLevel • TAMPCR TAMP3TRG LL_RTC_TAMPER_EnableActiveLevel

LL_RTC_TAMPER_DisableActiveLevel

Function name	**__STATIC_INLINE void LL_RTC_TAMPER_DisableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)**
Function description	Disable Active level for Tamper input.
Parameters	• **RTCx:** RTC Instance • **Tamper:** This parameter can be a combination of the following values: – LL_RTC_TAMPER_ACTIVELEVEL_TAMP1 – LL_RTC_TAMPER_ACTIVELEVEL_TAMP2 – LL_RTC_TAMPER_ACTIVELEVEL_TAMP3
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1TRG LL_RTC_TAMPER_DisableActiveLevel • TAMPCR TAMP2TRG LL_RTC_TAMPER_DisableActiveLevel • TAMPCR TAMP3TRG LL_RTC_TAMPER_DisableActiveLevel

LL_RTC_WAKEUP_Enable

Function name	**__STATIC_INLINE void LL_RTC_WAKEUP_Enable (RTC_TypeDef * RTCx)**

Function description	Enable Wakeup timer.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR WUTE LL_RTC_WAKEUP_Enable

LL_RTC_WAKEUP_Disable

Function name	**__STATIC_INLINE void LL_RTC_WAKEUP_Disable (RTC_TypeDef * RTCx)**
Function description	Disable Wakeup timer.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR WUTE LL_RTC_WAKEUP_Disable

LL_RTC_WAKEUP_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_RTC_WAKEUP_IsEnabled (RTC_TypeDef * RTCx)**
Function description	Check if Wakeup timer is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR WUTE LL_RTC_WAKEUP_IsEnabled

LL_RTC_WAKEUP_SetClock

Function name	**__STATIC_INLINE void LL_RTC_WAKEUP_SetClock (RTC_TypeDef * RTCx, uint32_t WakeupClock)**
Function description	Select Wakeup clock.
Parameters	• **RTCx:** RTC Instance • **WakeupClock:** This parameter can be one of the following values: – LL_RTC_WAKEUPCLOCK_DIV_16 – LL_RTC_WAKEUPCLOCK_DIV_8 – LL_RTC_WAKEUPCLOCK_DIV_4 – LL_RTC_WAKEUPCLOCK_DIV_2 – LL_RTC_WAKEUPCLOCK_CKSPRE

	– LL_RTC_WAKEUPCLOCK_CKSPRE_WUT
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • Bit can be written only when RTC_CR WUTE bit = 0 and RTC_ISR WUTWF bit = 1
Reference Manual to LL API cross reference:	• CR WUCKSEL LL_RTC_WAKEUP_SetClock

LL_RTC_WAKEUP_GetClock

Function name	**__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetClock (RTC_TypeDef * RTCx)**
Function description	Get Wakeup clock.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_WAKEUPCLOCK_DIV_16 – LL_RTC_WAKEUPCLOCK_DIV_8 – LL_RTC_WAKEUPCLOCK_DIV_4 – LL_RTC_WAKEUPCLOCK_DIV_2 – LL_RTC_WAKEUPCLOCK_CKSPRE – LL_RTC_WAKEUPCLOCK_CKSPRE_WUT
Reference Manual to LL API cross reference:	• CR WUCKSEL LL_RTC_WAKEUP_GetClock

LL_RTC_WAKEUP_SetAutoReload

Function name	**__STATIC_INLINE void LL_RTC_WAKEUP_SetAutoReload (RTC_TypeDef * RTCx, uint32_t Value)**
Function description	Set Wakeup auto-reload value.
Parameters	• **RTCx:** RTC Instance • **Value:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Return values	• **None:**
Notes	• Bit can be written only when WUTWF is set to 1 in RTC_ISR
Reference Manual to LL API cross reference:	• WUTR WUT LL_RTC_WAKEUP_SetAutoReload

LL_RTC_WAKEUP_GetAutoReload

Function name	**__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetAutoReload (RTC_TypeDef * RTCx)**
Function description	Get Wakeup auto-reload value.

Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• WUTR WUT LL_RTC_WAKEUP_GetAutoReload

LL_RTC_BAK_SetRegister

Function name	**__STATIC_INLINE void LL_RTC_BAK_SetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t Data)**
Function description	Writes a data in a specified RTC Backup data register.
Parameters	• **RTCx:** RTC Instance • **BackupRegister:** This parameter can be one of the following values: – LL_RTC_BKP_DR0 – LL_RTC_BKP_DR1 – LL_RTC_BKP_DR2 – LL_RTC_BKP_DR3 – LL_RTC_BKP_DR4 – LL_RTC_BKP_DR5 – LL_RTC_BKP_DR6 – LL_RTC_BKP_DR7 – LL_RTC_BKP_DR8 – LL_RTC_BKP_DR9 – LL_RTC_BKP_DR10 – LL_RTC_BKP_DR11 – LL_RTC_BKP_DR12 – LL_RTC_BKP_DR13 – LL_RTC_BKP_DR14 – LL_RTC_BKP_DR15 – LL_RTC_BKP_DR16 – LL_RTC_BKP_DR17 – LL_RTC_BKP_DR18 – LL_RTC_BKP_DR19 – LL_RTC_BKP_DR20 – LL_RTC_BKP_DR21 – LL_RTC_BKP_DR22 – LL_RTC_BKP_DR23 – LL_RTC_BKP_DR24 – LL_RTC_BKP_DR25 – LL_RTC_BKP_DR26 – LL_RTC_BKP_DR27 – LL_RTC_BKP_DR28 – LL_RTC_BKP_DR29 – LL_RTC_BKP_DR30 – LL_RTC_BKP_DR31 • **Data:** Value between Min_Data=0x00 and Max_Data=0xFFFFFFFF
Return values	• **None:**

Reference Manual to LL API cross reference:	• BKPxR BKP LL_RTC_BAK_SetRegister

LL_RTC_BAK_GetRegister

Function name	**__STATIC_INLINE uint32_t LL_RTC_BAK_GetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister)**
Function description	Reads data from the specified RTC Backup data Register.
Parameters	• **RTCx:** RTC Instance • **BackupRegister:** This parameter can be one of the following values: – LL_RTC_BKP_DR0 – LL_RTC_BKP_DR1 – LL_RTC_BKP_DR2 – LL_RTC_BKP_DR3 – LL_RTC_BKP_DR4 – LL_RTC_BKP_DR5 – LL_RTC_BKP_DR6 – LL_RTC_BKP_DR7 – LL_RTC_BKP_DR8 – LL_RTC_BKP_DR9 – LL_RTC_BKP_DR10 – LL_RTC_BKP_DR11 – LL_RTC_BKP_DR12 – LL_RTC_BKP_DR13 – LL_RTC_BKP_DR14 – LL_RTC_BKP_DR15 – LL_RTC_BKP_DR16 – LL_RTC_BKP_DR17 – LL_RTC_BKP_DR18 – LL_RTC_BKP_DR19 – LL_RTC_BKP_DR20 – LL_RTC_BKP_DR21 – LL_RTC_BKP_DR22 – LL_RTC_BKP_DR23 – LL_RTC_BKP_DR24 – LL_RTC_BKP_DR25 – LL_RTC_BKP_DR26 – LL_RTC_BKP_DR27 – LL_RTC_BKP_DR28 – LL_RTC_BKP_DR29 – LL_RTC_BKP_DR30 – LL_RTC_BKP_DR31
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFFFFFFFF
Reference Manual to LL API cross reference:	• BKPxR BKP LL_RTC_BAK_GetRegister

LL_RTC_CAL_SetOutputFreq

Function name	**__STATIC_INLINE void LL_RTC_CAL_SetOutputFreq (RTC_TypeDef * RTCx, uint32_t Frequency)**
Function description	Set Calibration output frequency (1 Hz or 512 Hz)
Parameters	• **RTCx:** RTC Instance • **Frequency:** This parameter can be one of the following values: – LL_RTC_CALIB_OUTPUT_NONE – LL_RTC_CALIB_OUTPUT_1HZ – LL_RTC_CALIB_OUTPUT_512HZ
Return values	• **None:**
Notes	• Bits are write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR COE LL_RTC_CAL_SetOutputFreq • CR COSEL LL_RTC_CAL_SetOutputFreq

LL_RTC_CAL_GetOutputFreq

Function name	**__STATIC_INLINE uint32_t LL_RTC_CAL_GetOutputFreq (RTC_TypeDef * RTCx)**
Function description	Get Calibration output frequency (1 Hz or 512 Hz)
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_CALIB_OUTPUT_NONE – LL_RTC_CALIB_OUTPUT_1HZ – LL_RTC_CALIB_OUTPUT_512HZ
Reference Manual to LL API cross reference:	• CR COE LL_RTC_CAL_GetOutputFreq • CR COSEL LL_RTC_CAL_GetOutputFreq

LL_RTC_CAL_SetPulse

Function name	**__STATIC_INLINE void LL_RTC_CAL_SetPulse (RTC_TypeDef * RTCx, uint32_t Pulse)**
Function description	Insert or not One RTCCLK pulse every 2exp11 pulses (frequency increased by 488.5 ppm)
Parameters	• **RTCx:** RTC Instance • **Pulse:** This parameter can be one of the following values: – LL_RTC_CALIB_INSERTPULSE_NONE – LL_RTC_CALIB_INSERTPULSE_SET
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • Bit can be written only when RECALPF is set to 0 in RTC_ISR

Reference Manual to LL API cross reference:	• CALR CALP LL_RTC_CAL_SetPulse

LL_RTC_CAL_IsPulseInserted

Function name	**__STATIC_INLINE uint32_t LL_RTC_CAL_IsPulseInserted (RTC_TypeDef * RTCx)**
Function description	Check if one RTCCLK has been inserted or not every 2exp11 pulses (frequency increased by 488.5 ppm)
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CALR CALP LL_RTC_CAL_IsPulseInserted

LL_RTC_CAL_SetPeriod

Function name	**__STATIC_INLINE void LL_RTC_CAL_SetPeriod (RTC_TypeDef * RTCx, uint32_t Period)**
Function description	Set the calibration cycle period.
Parameters	• **RTCx:** RTC Instance • **Period:** This parameter can be one of the following values: – LL_RTC_CALIB_PERIOD_32SEC – LL_RTC_CALIB_PERIOD_16SEC – LL_RTC_CALIB_PERIOD_8SEC
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • Bit can be written only when RECALPF is set to 0 in RTC_ISR
Reference Manual to LL API cross reference:	• CALR CALW8 LL_RTC_CAL_SetPeriod • CALR CALW16 LL_RTC_CAL_SetPeriod

LL_RTC_CAL_GetPeriod

Function name	**__STATIC_INLINE uint32_t LL_RTC_CAL_GetPeriod (RTC_TypeDef * RTCx)**
Function description	Get the calibration cycle period.
Parameters	• **RTCx:** RTC Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_CALIB_PERIOD_32SEC – LL_RTC_CALIB_PERIOD_16SEC – LL_RTC_CALIB_PERIOD_8SEC
Reference Manual to LL API cross	• CALR CALW8 LL_RTC_CAL_GetPeriod • CALR CALW16 LL_RTC_CAL_GetPeriod

reference:

LL_RTC_CAL_SetMinus

Function name	**__STATIC_INLINE void LL_RTC_CAL_SetMinus (RTC_TypeDef * RTCx, uint32_t CalibMinus)**
Function description	Set Calibration minus.
Parameters	• **RTCx:** RTC Instance • **CalibMinus:** Value between Min_Data=0x00 and Max_Data=0x1FF
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • Bit can be written only when RECALPF is set to 0 in RTC_ISR
Reference Manual to LL API cross reference:	• CALR CALM LL_RTC_CAL_SetMinus

LL_RTC_CAL_GetMinus

Function name	**__STATIC_INLINE uint32_t LL_RTC_CAL_GetMinus (RTC_TypeDef * RTCx)**
Function description	Get Calibration minus.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data= 0x1FF
Reference Manual to LL API cross reference:	• CALR CALM LL_RTC_CAL_GetMinus

LL_RTC_IsActiveFlag_ITS

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITS (RTC_TypeDef * RTCx)**
Function description	Get Internal Time-stamp flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ITSF LL_RTC_IsActiveFlag_ITS

LL_RTC_IsActiveFlag_RECALP

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RECALP (RTC_TypeDef * RTCx)**
Function description	Get Recalibration pending Flag.

Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RECALPF LL_RTC_IsActiveFlag_RECALP

LL_RTC_IsActiveFlag_TAMP3

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3 (RTC_TypeDef * RTCx)**
Function description	Get RTC_TAMP3 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TAMP3F LL_RTC_IsActiveFlag_TAMP3

LL_RTC_IsActiveFlag_TAMP2

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2 (RTC_TypeDef * RTCx)**
Function description	Get RTC_TAMP2 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TAMP2F LL_RTC_IsActiveFlag_TAMP2

LL_RTC_IsActiveFlag_TAMP1

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1 (RTC_TypeDef * RTCx)**
Function description	Get RTC_TAMP1 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TAMP1F LL_RTC_IsActiveFlag_TAMP1

LL_RTC_IsActiveFlag_TSOV

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOV (RTC_TypeDef * RTCx)**
Function description	Get Time-stamp overflow flag.
Parameters	• **RTCx:** RTC Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TSOVF LL_RTC_IsActiveFlag_TSOV

LL_RTC_IsActiveFlag_TS

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TS (RTC_TypeDef * RTCx)**
Function description	Get Time-stamp flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TSF LL_RTC_IsActiveFlag_TS

LL_RTC_IsActiveFlag_WUT

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUT (RTC_TypeDef * RTCx)**
Function description	Get Wakeup timer flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR WUTF LL_RTC_IsActiveFlag_WUT

LL_RTC_IsActiveFlag_ALRB

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRB (RTC_TypeDef * RTCx)**
Function description	Get Alarm B flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ALRBF LL_RTC_IsActiveFlag_ALRB

LL_RTC_IsActiveFlag_ALRA

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRA (RTC_TypeDef * RTCx)**
Function description	Get Alarm A flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• ISR ALRAF LL_RTC_IsActiveFlag_ALRA

LL_RTC_ClearFlag_ITS

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_ITS (RTC_TypeDef * RTCx)**
Function description	Clear Internal Time-stamp flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR ITSF LL_RTC_ClearFlag_ITS

LL_RTC_ClearFlag_TAMP3

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TAMP3 (RTC_TypeDef * RTCx)**
Function description	Clear RTC_TAMP3 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR TAMP3F LL_RTC_ClearFlag_TAMP3

LL_RTC_ClearFlag_TAMP2

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TAMP2 (RTC_TypeDef * RTCx)**
Function description	Clear RTC_TAMP2 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR TAMP2F LL_RTC_ClearFlag_TAMP2

LL_RTC_ClearFlag_TAMP1

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TAMP1 (RTC_TypeDef * RTCx)**
Function description	Clear RTC_TAMP1 detection flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross	• ISR TAMP1F LL_RTC_ClearFlag_TAMP1

reference:

LL_RTC_ClearFlag_TSOV

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TSOV (RTC_TypeDef * RTCx)**
Function description	Clear Time-stamp overflow flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR TSOVF LL_RTC_ClearFlag_TSOV

LL_RTC_ClearFlag_TS

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TS (RTC_TypeDef * RTCx)**
Function description	Clear Time-stamp flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR TSF LL_RTC_ClearFlag_TS

LL_RTC_ClearFlag_WUT

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_WUT (RTC_TypeDef * RTCx)**
Function description	Clear Wakeup timer flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR WUTF LL_RTC_ClearFlag_WUT

LL_RTC_ClearFlag_ALRB

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_ALRB (RTC_TypeDef * RTCx)**
Function description	Clear Alarm B flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR ALRBF LL_RTC_ClearFlag_ALRB

LL_RTC_ClearFlag_ALRA

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_ALRA (RTC_TypeDef * RTCx)**
Function description	Clear Alarm A flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR ALRAF LL_RTC_ClearFlag_ALRA

LL_RTC_IsActiveFlag_INIT

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INIT (RTC_TypeDef * RTCx)**
Function description	Get Initialization flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR INITF LL_RTC_IsActiveFlag_INIT

LL_RTC_IsActiveFlag_RS

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (RTC_TypeDef * RTCx)**
Function description	Get Registers synchronization flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RSF LL_RTC_IsActiveFlag_RS

LL_RTC_ClearFlag_RS

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_RS (RTC_TypeDef * RTCx)**
Function description	Clear Registers synchronization flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ISR RSF LL_RTC_ClearFlag_RS

LL_RTC_IsActiveFlag_INITS

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INITS (RTC_TypeDef * RTCx)**
Function description	Get Initialization status flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR INITS LL_RTC_IsActiveFlag_INITS

LL_RTC_IsActiveFlag_SHP

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SHP (RTC_TypeDef * RTCx)**
Function description	Get Shift operation pending flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SHPF LL_RTC_IsActiveFlag_SHP

LL_RTC_IsActiveFlag_WUTW

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTW (RTC_TypeDef * RTCx)**
Function description	Get Wakeup timer write flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR WUTWF LL_RTC_IsActiveFlag_WUTW

LL_RTC_IsActiveFlag_ALRBW

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRBW (RTC_TypeDef * RTCx)**
Function description	Get Alarm B write flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ALRBWF LL_RTC_IsActiveFlag_ALRBW

LL_RTC_IsActiveFlag_ALRAW

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAW (RTC_TypeDef * RTCx)**
Function description	Get Alarm A write flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ALRAWF LL_RTC_IsActiveFlag_ALRAW

LL_RTC_EnableIT_TS

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TS (RTC_TypeDef * RTCx)**
Function description	Enable Time-stamp interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR TSIE LL_RTC_EnableIT_TS

LL_RTC_DisableIT_TS

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TS (RTC_TypeDef * RTCx)**
Function description	Disable Time-stamp interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR TSIE LL_RTC_DisableIT_TS

LL_RTC_EnableIT_WUT

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_WUT (RTC_TypeDef * RTCx)**
Function description	Enable Wakeup timer interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection

	function should be called before.
Reference Manual to LL API cross reference:	• CR WUTIE LL_RTC_EnableIT_WUT

LL_RTC_DisableIT_WUT

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_WUT (RTC_TypeDef * RTCx)**
Function description	Disable Wakeup timer interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR WUTIE LL_RTC_DisableIT_WUT

LL_RTC_EnableIT_ALRB

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_ALRB (RTC_TypeDef * RTCx)**
Function description	Enable Alarm B interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRBIE LL_RTC_EnableIT_ALRB

LL_RTC_DisableIT_ALRB

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_ALRB (RTC_TypeDef * RTCx)**
Function description	Disable Alarm B interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRBIE LL_RTC_DisableIT_ALRB

LL_RTC_EnableIT_ALRA

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_ALRA (RTC_TypeDef * RTCx)**
Function description	Enable Alarm A interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRAIE LL_RTC_EnableIT_ALRA

LL_RTC_DisableIT_ALRA

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_ALRA (RTC_TypeDef * RTCx)**
Function description	Disable Alarm A interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CR ALRAIE LL_RTC_DisableIT_ALRA

LL_RTC_EnableIT_TAMP3

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TAMP3 (RTC_TypeDef * RTCx)**
Function description	Enable Tamper 3 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP3IE LL_RTC_EnableIT_TAMP3

LL_RTC_DisableIT_TAMP3

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TAMP3 (RTC_TypeDef * RTCx)**
Function description	Disable Tamper 3 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to	• TAMPCR TAMP3IE LL_RTC_DisableIT_TAMP3

LL API cross reference:	

LL_RTC_EnableIT_TAMP2

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TAMP2 (RTC_TypeDef * RTCx)**
Function description	Enable Tamper 2 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP2IE LL_RTC_EnableIT_TAMP2

LL_RTC_DisableIT_TAMP2

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TAMP2 (RTC_TypeDef * RTCx)**
Function description	Disable Tamper 2 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP2IE LL_RTC_DisableIT_TAMP2

LL_RTC_EnableIT_TAMP1

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TAMP1 (RTC_TypeDef * RTCx)**
Function description	Enable Tamper 1 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMP1IE LL_RTC_EnableIT_TAMP1

LL_RTC_DisableIT_TAMP1

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TAMP1 (RTC_TypeDef * RTCx)**
Function description	Disable Tamper 1 interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross	• TAMPCR TAMP1IE LL_RTC_DisableIT_TAMP1

reference:

LL_RTC_EnableIT_TAMP

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TAMP (RTC_TypeDef * RTCx)**
Function description	Enable all Tamper Interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMPIE LL_RTC_EnableIT_TAMP

LL_RTC_DisableIT_TAMP

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TAMP (RTC_TypeDef * RTCx)**
Function description	Disable all Tamper Interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• TAMPCR TAMPIE LL_RTC_DisableIT_TAMP

LL_RTC_IsEnabledIT_TS

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TS (RTC_TypeDef * RTCx)**
Function description	Check if Time-stamp interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR TSIE LL_RTC_IsEnabledIT_TS

LL_RTC_IsEnabledIT_WUT

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_WUT (RTC_TypeDef * RTCx)**
Function description	Check if Wakeup timer interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR WUTIE LL_RTC_IsEnabledIT_WUT

LL_RTC_IsEnabledIT_ALRB

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRB (RTC_TypeDef * RTCx)**
Function description	Check if Alarm B interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR ALRBIE LL_RTC_IsEnabledIT_ALRB

LL_RTC_IsEnabledIT_ALRA

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRA (RTC_TypeDef * RTCx)**
Function description	Check if Alarm A interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR ALRAIE LL_RTC_IsEnabledIT_ALRA

LL_RTC_IsEnabledIT_TAMP3

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP3 (RTC_TypeDef * RTCx)**
Function description	Check if Tamper 3 interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• TAMPCR TAMP3IE LL_RTC_IsEnabledIT_TAMP3

LL_RTC_IsEnabledIT_TAMP2

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP2 (RTC_TypeDef * RTCx)**
Function description	Check if Tamper 2 interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• TAMPCR TAMP2IE LL_RTC_IsEnabledIT_TAMP2

LL_RTC_IsEnabledIT_TAMP1

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP1 (RTC_TypeDef * RTCx)**
Function description	Check if Tamper 1 interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• TAMPCR TAMP1IE LL_RTC_IsEnabledIT_TAMP1

LL_RTC_IsEnabledIT_TAMP

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP (RTC_TypeDef * RTCx)**
Function description	Check if all the TAMPER interrupts are enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• TAMPCR TAMPIE LL_RTC_IsEnabledIT_TAMP

LL_RTC_DeInit

Function name	**ErrorStatus LL_RTC_DeInit (RTC_TypeDef * RTCx)**
Function description	De-Initializes the RTC registers to their default reset values.
Parameters	• **RTCx:** RTC Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC registers are de-initialized – ERROR: RTC registers are not de-initialized
Notes	• This function doesn't reset the RTC Clock source and RTC Backup Data registers.

LL_RTC_Init

Function name	**ErrorStatus LL_RTC_Init (RTC_TypeDef * RTCx, LL_RTC_InitTypeDef * RTC_InitStruct)**
Function description	Initializes the RTC registers according to the specified parameters in RTC_InitStruct.
Parameters	• **RTCx:** RTC Instance • **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC registers are initialized – ERROR: RTC registers are not initialized

Notes	• The RTC Prescaler register is write protected and can be written in initialization mode only.

LL_RTC_StructInit

Function name	**void LL_RTC_StructInit (LL_RTC_InitTypeDef * RTC_InitStruct)**
Function description	Set each LL_RTC_InitTypeDef field to default value.
Parameters	• **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure which will be initialized.
Return values	• **None:**

LL_RTC_TIME_Init

Function name	**ErrorStatus LL_RTC_TIME_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef * RTC_TimeStruct)**
Function description	Set the RTC current time.
Parameters	• **RTCx:** RTC Instance • **RTC_Format:** This parameter can be one of the following values: – LL_RTC_FORMAT_BIN – LL_RTC_FORMAT_BCD • **RTC_TimeStruct:** pointer to a RTC_TimeTypeDef structure that contains the time configuration information for the RTC.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC Time register is configured – ERROR: RTC Time register is not configured

LL_RTC_TIME_StructInit

Function name	**void LL_RTC_TIME_StructInit (LL_RTC_TimeTypeDef * RTC_TimeStruct)**
Function description	Set each LL_RTC_TimeTypeDef field to default value (Time = 00h:00min:00sec).
Parameters	• **RTC_TimeStruct:** pointer to a LL_RTC_TimeTypeDef structure which will be initialized.
Return values	• **None:**

LL_RTC_DATE_Init

Function name	**ErrorStatus LL_RTC_DATE_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef * RTC_DateStruct)**
Function description	Set the RTC current date.
Parameters	• **RTCx:** RTC Instance • **RTC_Format:** This parameter can be one of the following values:

- LL_RTC_FORMAT_BIN
- LL_RTC_FORMAT_BCD

- **RTC_DateStruct:** pointer to a RTC_DateTypeDef structure that contains the date configuration information for the RTC.

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Day register is configured
  - ERROR: RTC Day register is not configured

LL_RTC_DATE_StructInit

Function name	**void LL_RTC_DATE_StructInit (LL_RTC_DateTypeDef * RTC_DateStruct)**
Function description	Set each LL_RTC_DateTypeDef field to default value (date = Monday, January 01 xx00)
Parameters	• **RTC_DateStruct:** pointer to a LL_RTC_DateTypeDef structure which will be initialized.
Return values	• **None:**

LL_RTC_ALMA_Init

Function name	**ErrorStatus LL_RTC_ALMA_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)**
Function description	Set the RTC Alarm A.
Parameters	• **RTCx:** RTC Instance • **RTC_Format:** This parameter can be one of the following values: – LL_RTC_FORMAT_BIN – LL_RTC_FORMAT_BCD • **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ALARMA registers are configured – ERROR: ALARMA registers are not configured
Notes	• The Alarm register can only be written when the corresponding Alarm is disabled (Use LL_RTC_ALMA_Disable function).

LL_RTC_ALMB_Init

Function name	**ErrorStatus LL_RTC_ALMB_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)**
Function description	Set the RTC Alarm B.
Parameters	• **RTCx:** RTC Instance • **RTC_Format:** This parameter can be one of the following values: – LL_RTC_FORMAT_BIN

- LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: ALARMB registers are configured
  - ERROR: ALARMB registers are not configured

Notes
- The Alarm register can only be written when the corresponding Alarm is disabled (LL_RTC_ALMB_Disable function).

LL_RTC_ALMA_StructInit

Function name
void LL_RTC_ALMA_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description
Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values
- **None:**

LL_RTC_ALMB_StructInit

Function name
void LL_RTC_ALMB_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description
Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values
- **None:**

LL_RTC_EnterInitMode

Function name
ErrorStatus LL_RTC_EnterInitMode (RTC_TypeDef * RTCx)

Function description
Enters the RTC Initialization mode.

Parameters
- **RTCx:** RTC Instance

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC is in Init mode
  - ERROR: RTC is not in Init mode

Notes
- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_ExitInitMode

Function name
ErrorStatus LL_RTC_ExitInitMode (RTC_TypeDef * RTCx)

Function description
Exit the RTC Initialization mode.

Parameters	• **RTCx:** RTC Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC exited from in Init mode – ERROR: Not applicable
Notes	• When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles. • The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_WaitForSynchro

Function name	**ErrorStatus LL_RTC_WaitForSynchro (RTC_TypeDef * RTCx)**
Function description	Waits until the RTC Time and Day registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	• **RTCx:** RTC Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC registers are synchronised – ERROR: RTC registers are not synchronised
Notes	• The RTC Resynchronization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function. • To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

93.3 RTC Firmware driver defines

93.3.1 RTC

ALARM OUTPUT

LL_RTC_ALARMOUT_DISABLE	Output disabled
LL_RTC_ALARMOUT_ALMA	Alarm A output enabled
LL_RTC_ALARMOUT_ALMB	Alarm B output enabled
LL_RTC_ALARMOUT_WAKEUP	Wakeup output enabled

ALARM OUTPUT TYPE

LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN	RTC_ALARM, when mapped on PC13, is open-drain output
LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL	RTC_ALARM, when mapped on PC13, is push-pull output

ALARMA MASK

LL_RTC_ALMA_MASK_NONE	No masks applied on Alarm A
LL_RTC_ALMA_MASK_DATEWEEKDAY	Date/day do not care in Alarm A comparison

LL_RTC_ALMA_MASK_HOURS	Hours do not care in Alarm A comparison
LL_RTC_ALMA_MASK_MINUTES	Minutes do not care in Alarm A comparison
LL_RTC_ALMA_MASK_SECONDS	Seconds do not care in Alarm A comparison
LL_RTC_ALMA_MASK_ALL	Masks all

ALARMA TIME FORMAT

LL_RTC_ALMA_TIME_FORMAT_AM	AM or 24-hour format
LL_RTC_ALMA_TIME_FORMAT_PM	PM

RTC Alarm A Date WeekDay

LL_RTC_ALMA_DATEWEEKDAYSEL_DATE	Alarm A Date is selected
LL_RTC_ALMA_DATEWEEKDAYSEL_WEEKDAY	Alarm A WeekDay is selected

ALARMB MASK

LL_RTC_ALMB_MASK_NONE	No masks applied on Alarm B
LL_RTC_ALMB_MASK_DATEWEEKDAY	Date/day do not care in Alarm B comparison
LL_RTC_ALMB_MASK_HOURS	Hours do not care in Alarm B comparison
LL_RTC_ALMB_MASK_MINUTES	Minutes do not care in Alarm B comparison
LL_RTC_ALMB_MASK_SECONDS	Seconds do not care in Alarm B comparison
LL_RTC_ALMB_MASK_ALL	Masks all

ALARMB TIME FORMAT

LL_RTC_ALMB_TIME_FORMAT_AM	AM or 24-hour format
LL_RTC_ALMB_TIME_FORMAT_PM	PM

RTC Alarm B Date WeekDay

LL_RTC_ALMB_DATEWEEKDAYSEL_DATE	Alarm B Date is selected
LL_RTC_ALMB_DATEWEEKDAYSEL_WEEKDAY	Alarm B WeekDay is selected

BACKUP

LL_RTC_BKP_DR0

LL_RTC_BKP_DR1

LL_RTC_BKP_DR2

LL_RTC_BKP_DR3

LL_RTC_BKP_DR4

LL_RTC_BKP_DR5

LL_RTC_BKP_DR6

LL_RTC_BKP_DR7

LL_RTC_BKP_DR8

LL_RTC_BKP_DR9

LL_RTC_BKP_DR10

LL_RTC_BKP_DR11

LL_RTC_BKP_DR12

LL_RTC_BKP_DR13

LL_RTC_BKP_DR14

LL_RTC_BKP_DR15

LL_RTC_BKP_DR16

LL_RTC_BKP_DR17

LL_RTC_BKP_DR18

LL_RTC_BKP_DR19

LL_RTC_BKP_DR20

LL_RTC_BKP_DR21

LL_RTC_BKP_DR22

LL_RTC_BKP_DR23

LL_RTC_BKP_DR24

LL_RTC_BKP_DR25

LL_RTC_BKP_DR26

LL_RTC_BKP_DR27

LL_RTC_BKP_DR28

LL_RTC_BKP_DR29

LL_RTC_BKP_DR30

LL_RTC_BKP_DR31

Calibration pulse insertion

LL_RTC_CALIB_INSERTPULSE_NONE	No RTCCLK pulses are added
LL_RTC_CALIB_INSERTPULSE_SET	One RTCCLK pulse is effectively inserted every 2exp11 pulses (frequency increased by 488.5 ppm)

Calibration output

LL_RTC_CALIB_OUTPUT_NONE	Calibration output disabled
LL_RTC_CALIB_OUTPUT_1HZ	Calibration output is 1 Hz
LL_RTC_CALIB_OUTPUT_512HZ	Calibration output is 512 Hz

Calibration period

LL_RTC_CALIB_PERIOD_32SEC	Use a 32-second calibration cycle period
LL_RTC_CALIB_PERIOD_16SEC	Use a 16-second calibration cycle period
LL_RTC_CALIB_PERIOD_8SEC	Use a 8-second calibration cycle period

FORMAT

LL_RTC_FORMAT_BIN	Binary data format
LL_RTC_FORMAT_BCD	BCD data format

Get Flags Defines

LL_RTC_ISR_ITSF

LL_RTC_ISR_RECALPF

LL_RTC_ISR_TAMP3F

LL_RTC_ISR_TAMP2F

LL_RTC_ISR_TAMP1F

LL_RTC_ISR_TSOVF

LL_RTC_ISR_TSF

LL_RTC_ISR_WUTF

LL_RTC_ISR_ALRBF

LL_RTC_ISR_ALRAF

LL_RTC_ISR_INITF

LL_RTC_ISR_RSF

LL_RTC_ISR_INITS

LL_RTC_ISR_SHPF

LL_RTC_ISR_WUTWF

LL_RTC_ISR_ALRBWF

LL_RTC_ISR_ALRAWF

HOUR FORMAT

LL_RTC_HOURFORMAT_24HOUR	24 hour/day format
LL_RTC_HOURFORMAT_AMPM	AM/PM hour format

IT Defines

LL_RTC_CR_TSIE

LL_RTC_CR_WUTIE

LL_RTC_CR_ALRBIE

LL_RTC_CR_ALRAIE

LL_RTC_TAMPCR_TAMP3IE

LL_RTC_TAMPCR_TAMP2IE

LL_RTC_TAMPCR_TAMP1IE

LL_RTC_TAMPCR_TAMPIE

MONTH

LL_RTC_MONTH_JANUARY	January
LL_RTC_MONTH_FEBRUARY	February
LL_RTC_MONTH_MARCH	March
LL_RTC_MONTH_APRIL	April
LL_RTC_MONTH_MAY	May
LL_RTC_MONTH_JUNE	June

LL_RTC_MONTH_JULY	July
LL_RTC_MONTH_AUGUST	August
LL_RTC_MONTH_SEPTEMBER	September
LL_RTC_MONTH_OCTOBER	October
LL_RTC_MONTH_NOVEMBER	November
LL_RTC_MONTH_DECEMBER	December

OUTPUT POLARITY PIN

LL_RTC_OUTPUTPOLARITY_PIN_HIGH	Pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)
LL_RTC_OUTPUTPOLARITY_PIN_LOW	Pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

SHIFT SECOND

LL_RTC_SHIFT_SECOND_DELAY

LL_RTC_SHIFT_SECOND_ADVANCE

TAMPER

LL_RTC_TAMPER_1	RTC_TAMP1 input detection
LL_RTC_TAMPER_2	RTC_TAMP2 input detection
LL_RTC_TAMPER_3	RTC_TAMP3 input detection

TAMPER ACTIVE LEVEL

LL_RTC_TAMPER_ACTIVELEVEL_TAMP1	RTC_TAMP1 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event
LL_RTC_TAMPER_ACTIVELEVEL_TAMP2	RTC_TAMP2 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event
LL_RTC_TAMPER_ACTIVELEVEL_TAMP3	RTC_TAMP3 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

TAMPER DURATION

LL_RTC_TAMPER_DURATION_1RTCCLK	Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
LL_RTC_TAMPER_DURATION_2RTCCLK	Tamper pins are pre-charged before sampling during 2 RTCCLK cycles
LL_RTC_TAMPER_DURATION_4RTCCLK	Tamper pins are pre-charged before sampling during 4 RTCCLK cycles
LL_RTC_TAMPER_DURATION_8RTCCLK	Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

TAMPER FILTER

LL_RTC_TAMPER_FILTER_DISABLE	Tamper filter is disabled
LL_RTC_TAMPER_FILTER_2SAMPLE	Tamper is activated after 2 consecutive samples at the active level
LL_RTC_TAMPER_FILTER_4SAMPLE	Tamper is activated after 4 consecutive samples at the active level
LL_RTC_TAMPER_FILTER_8SAMPLE	Tamper is activated after 8 consecutive samples at the active level.

TAMPER MASK

LL_RTC_TAMPER_MASK_TAMPER1	Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware.The backup registers are not erased
LL_RTC_TAMPER_MASK_TAMPER2	Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.
LL_RTC_TAMPER_MASK_TAMPER3	Tamper 3 event generates a trigger event. TAMP3F is masked and internally cleared by hardware. The backup registers are not erased

TAMPER NO ERASE

LL_RTC_TAMPER_NOERASE_TAMPER1	Tamper 1 event does not erase the backup registers.
LL_RTC_TAMPER_NOERASE_TAMPER2	Tamper 2 event does not erase the backup registers.
LL_RTC_TAMPER_NOERASE_TAMPER3	Tamper 3 event does not erase the backup registers.

TAMPER SAMPLING FREQUENCY DIVIDER

LL_RTC_TAMPER_SAMPLFREQDIV_32768	Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
LL_RTC_TAMPER_SAMPLFREQDIV_16384	Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
LL_RTC_TAMPER_SAMPLFREQDIV_8192	Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192
LL_RTC_TAMPER_SAMPLFREQDIV_4096	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
LL_RTC_TAMPER_SAMPLFREQDIV_2048	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
LL_RTC_TAMPER_SAMPLFREQDIV_1024	Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
LL_RTC_TAMPER_SAMPLFREQDIV_512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
LL_RTC_TAMPER_SAMPLFREQDIV_256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

TIMESTAMP EDGE

LL_RTC_TIMESTAMP_EDGE_RISING	RTC_TS input rising edge generates a time-stamp event

LL_RTC_TIMESTAMP_EDGE_FALLING	RTC_TS input falling edge generates a time-stamp even

TIME FORMAT

LL_RTC_TIME_FORMAT_AM_OR_24	AM or 24-hour format
LL_RTC_TIME_FORMAT_PM	PM

TIMESTAMP TIME FORMAT

LL_RTC_TS_TIME_FORMAT_AM	AM or 24-hour format
LL_RTC_TS_TIME_FORMAT_PM	PM

WAKEUP CLOCK DIV

LL_RTC_WAKEUPCLOCK_DIV_16	RTC/16 clock is selected
LL_RTC_WAKEUPCLOCK_DIV_8	RTC/8 clock is selected
LL_RTC_WAKEUPCLOCK_DIV_4	RTC/4 clock is selected
LL_RTC_WAKEUPCLOCK_DIV_2	RTC/2 clock is selected
LL_RTC_WAKEUPCLOCK_CKSPRE	ck_spre (usually 1 Hz) clock is selected
LL_RTC_WAKEUPCLOCK_CKSPRE_WUT	ck_spre (usually 1 Hz) clock is selected and 2exp16 is added to the WUT counter value

WEEK DAY

LL_RTC_WEEKDAY_MONDAY	Monday
LL_RTC_WEEKDAY_TUESDAY	Tuesday
LL_RTC_WEEKDAY_WEDNESDAY	Wednesday
LL_RTC_WEEKDAY_THURSDAY	Thrusday
LL_RTC_WEEKDAY_FRIDAY	Friday
LL_RTC_WEEKDAY_SATURDAY	Saturday
LL_RTC_WEEKDAY_SUNDAY	Sunday

Convert helper Macros

__LL_RTC_CONVERT_BIN2BCD

Description:

- Helper macro to convert a value from 2 digit decimal format to BCD format.

Parameters:

- __VALUE__: Byte to be converted

Return value:

- Converted: byte

__LL_RTC_CONVERT_BCD2BIN

Description:

- Helper macro to convert a value from BCD format to 2 digit decimal format.

Parameters:

- __VALUE__: BCD value to be converted

Return value:

- Converted: byte

Date helper Macros

__LL_RTC_GET_WEEKDAY

Description:

- Helper macro to retrieve weekday.

Parameters:

- __RTC_DATE__: Date returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

__LL_RTC_GET_YEAR

Description:

- Helper macro to retrieve Year in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Year: in BCD format (0x00 . . . 0x99)

__LL_RTC_GET_MONTH

Description:

- Helper macro to retrieve Month in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

__LL_RTC_GET_DAY

Description:

- Helper macro to retrieve Day in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Day: in BCD format (0x01 . . . 0x31)

Time helper Macros

__LL_RTC_GET_HOUR

Description:

- Helper macro to retrieve hour in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Hours: in BCD format (0x01. . .0x12 or between Min_Data=0x00 and Max_Data=0x23)

__LL_RTC_GET_MINUTE

Description:

- Helper macro to retrieve minute in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Minutes: in BCD format (0x00. . .0x59)

__LL_RTC_GET_SECOND

Description:

- Helper macro to retrieve second in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Seconds: in format (0x00. . .0x59)

Common Write and read registers Macros

LL_RTC_WriteReg

Description:

- Write a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RTC_ReadReg

Description:

- Read a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance

- __REG__: Register to be read

Return value:

- Register: value

94 LL SPI Generic Driver

94.1 SPI Firmware driver registers structures

94.1.1 LL_SPI_InitTypeDef

Data Fields

- ***uint32_t TransferDirection***
- ***uint32_t Mode***
- ***uint32_t DataWidth***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRate***
- ***uint32_t BitOrder***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPoly***

Field Documentation

- ***uint32_t LL_SPI_InitTypeDef::TransferDirection***
  Specifies the SPI unidirectional or bidirectional data mode. This parameter can be a value of ***SPI_LL_EC_TRANSFER_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferDirection()**.
- ***uint32_t LL_SPI_InitTypeDef::Mode***
  Specifies the SPI mode (Master/Slave). This parameter can be a value of ***SPI_LL_EC_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetMode()**.
- ***uint32_t LL_SPI_InitTypeDef::DataWidth***
  Specifies the SPI data width. This parameter can be a value of ***SPI_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_SPI_SetDataWidth()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_LL_EC_POLARITY***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPolarity()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_LL_EC_PHASE***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPhase()**.
- ***uint32_t LL_SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_LL_EC_NSS_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetNSSMode()**.
- ***uint32_t LL_SPI_InitTypeDef::BaudRate***
  Specifies the BaudRate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_LL_EC_BAUDRATEPRESCALER***.
  Note: The communication clock is derived from the master clock. The slave clock does not need to be set. This feature can be modified afterwards using unitary function **LL_SPI_SetBaudRatePrescaler()**.

- ***uint32_t LL_SPI_InitTypeDef::BitOrder***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_LL_EC_BIT_ORDER***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferBitOrder()**.
- ***uint32_t LL_SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_LL_EC_CRC_CALCULATION***.This feature can be modified afterwards using unitary functions **LL_SPI_EnableCRC()** and **LL_SPI_DisableCRC()**.
- ***uint32_t LL_SPI_InitTypeDef::CRCPoly***
  Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function **LL_SPI_SetCRCPolynomial()**.

94.2 SPI Firmware driver API description

94.2.1 Detailed description of functions

LL_SPI_Enable

Function name	**__STATIC_INLINE void LL_SPI_Enable (SPI_TypeDef * SPIx)**
Function description	Enable SPI peripheral.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 SPE LL_SPI_Enable

LL_SPI_Disable

Function name	**__STATIC_INLINE void LL_SPI_Disable (SPI_TypeDef * SPIx)**
Function description	Disable SPI peripheral.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• When disabling the SPI, follow the procedure described in the Reference Manual.
Reference Manual to LL API cross reference:	• CR1 SPE LL_SPI_Disable

LL_SPI_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabled (SPI_TypeDef * SPIx)**
Function description	Check if SPI peripheral is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CR1 SPE LL_SPI_IsEnabled

reference:

LL_SPI_SetMode

Function name	**__STATIC_INLINE void LL_SPI_SetMode (SPI_TypeDef * SPIx, uint32_t Mode)**
Function description	Set SPI operation mode to Master or Slave.
Parameters	• **SPIx:** SPI Instance • **Mode:** This parameter can be one of the following values: – LL_SPI_MODE_MASTER – LL_SPI_MODE_SLAVE
Return values	• **None:**
Notes	• This bit should not be changed when communication is ongoing.
Reference Manual to LL API cross reference:	• CR1 MSTR LL_SPI_SetMode • CR1 SSI LL_SPI_SetMode

LL_SPI_GetMode

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetMode (SPI_TypeDef * SPIx)**
Function description	Get SPI operation mode (Master or Slave)
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_MODE_MASTER – LL_SPI_MODE_SLAVE
Reference Manual to LL API cross reference:	• CR1 MSTR LL_SPI_GetMode • CR1 SSI LL_SPI_GetMode

LL_SPI_SetStandard

Function name	**__STATIC_INLINE void LL_SPI_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)**
Function description	Set serial protocol used.
Parameters	• **SPIx:** SPI Instance • **Standard:** This parameter can be one of the following values: – LL_SPI_PROTOCOL_MOTOROLA – LL_SPI_PROTOCOL_TI
Return values	• **None:**
Notes	• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross	• CR2 FRF LL_SPI_SetStandard

reference:

LL_SPI_GetStandard

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetStandard (SPI_TypeDef * SPIx)**
Function description	Get serial protocol used.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_PROTOCOL_MOTOROLA – LL_SPI_PROTOCOL_TI
Reference Manual to LL API cross reference:	• CR2 FRF LL_SPI_GetStandard

LL_SPI_SetClockPhase

Function name	**__STATIC_INLINE void LL_SPI_SetClockPhase (SPI_TypeDef * SPIx, uint32_t ClockPhase)**
Function description	Set clock phase.
Parameters	• **SPIx:** SPI Instance • **ClockPhase:** This parameter can be one of the following values: – LL_SPI_PHASE_1EDGE – LL_SPI_PHASE_2EDGE
Return values	• **None:**
Notes	• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR1 CPHA LL_SPI_SetClockPhase

LL_SPI_GetClockPhase

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetClockPhase (SPI_TypeDef * SPIx)**
Function description	Get clock phase.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_PHASE_1EDGE – LL_SPI_PHASE_2EDGE
Reference Manual to LL API cross reference:	• CR1 CPHA LL_SPI_GetClockPhase

LL_SPI_SetClockPolarity

Function name	**__STATIC_INLINE void LL_SPI_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)**
Function description	Set clock polarity.
Parameters	• **SPIx:** SPI Instance • **ClockPolarity:** This parameter can be one of the following values: – LL_SPI_POLARITY_LOW – LL_SPI_POLARITY_HIGH
Return values	• **None:**
Notes	• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR1 CPOL LL_SPI_SetClockPolarity

LL_SPI_GetClockPolarity

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetClockPolarity (SPI_TypeDef * SPIx)**
Function description	Get clock polarity.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_POLARITY_LOW – LL_SPI_POLARITY_HIGH
Reference Manual to LL API cross reference:	• CR1 CPOL LL_SPI_GetClockPolarity

LL_SPI_SetBaudRatePrescaler

Function name	**__STATIC_INLINE void LL_SPI_SetBaudRatePrescaler (SPI_TypeDef * SPIx, uint32_t BaudRate)**
Function description	Set baud rate prescaler.
Parameters	• **SPIx:** SPI Instance • **BaudRate:** This parameter can be one of the following values: – LL_SPI_BAUDRATEPRESCALER_DIV2 – LL_SPI_BAUDRATEPRESCALER_DIV4 – LL_SPI_BAUDRATEPRESCALER_DIV8 – LL_SPI_BAUDRATEPRESCALER_DIV16 – LL_SPI_BAUDRATEPRESCALER_DIV32 – LL_SPI_BAUDRATEPRESCALER_DIV64 – LL_SPI_BAUDRATEPRESCALER_DIV128 – LL_SPI_BAUDRATEPRESCALER_DIV256
Return values	• **None:**

Notes	• These bits should not be changed when communication is ongoing. SPI BaudRate = fPCLK/Prescaler.
Reference Manual to LL API cross reference:	• CR1 BR LL_SPI_SetBaudRatePrescaler

LL_SPI_GetBaudRatePrescaler

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetBaudRatePrescaler (SPI_TypeDef * SPIx)**
Function description	Get baud rate prescaler.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_BAUDRATEPRESCALER_DIV2 – LL_SPI_BAUDRATEPRESCALER_DIV4 – LL_SPI_BAUDRATEPRESCALER_DIV8 – LL_SPI_BAUDRATEPRESCALER_DIV16 – LL_SPI_BAUDRATEPRESCALER_DIV32 – LL_SPI_BAUDRATEPRESCALER_DIV64 – LL_SPI_BAUDRATEPRESCALER_DIV128 – LL_SPI_BAUDRATEPRESCALER_DIV256
Reference Manual to LL API cross reference:	• CR1 BR LL_SPI_GetBaudRatePrescaler

LL_SPI_SetTransferBitOrder

Function name	**__STATIC_INLINE void LL_SPI_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)**
Function description	Set transfer bit order.
Parameters	• **SPIx:** SPI Instance • **BitOrder:** This parameter can be one of the following values: – LL_SPI_LSB_FIRST – LL_SPI_MSB_FIRST
Return values	• **None:**
Notes	• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR1 LSBFIRST LL_SPI_SetTransferBitOrder

LL_SPI_GetTransferBitOrder

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetTransferBitOrder (SPI_TypeDef * SPIx)**
Function description	Get transfer bit order.
Parameters	• **SPIx:** SPI Instance

Return values	• **Returned:** value can be one of the following values: – LL_SPI_LSB_FIRST – LL_SPI_MSB_FIRST
Reference Manual to LL API cross reference:	• CR1 LSBFIRST LL_SPI_GetTransferBitOrder

LL_SPI_SetTransferDirection

Function name	**__STATIC_INLINE void LL_SPI_SetTransferDirection (SPI_TypeDef * SPIx, uint32_t TransferDirection)**
Function description	Set transfer direction mode.
Parameters	• **SPIx:** SPI Instance • **TransferDirection:** This parameter can be one of the following values: – LL_SPI_FULL_DUPLEX – LL_SPI_SIMPLEX_RX – LL_SPI_HALF_DUPLEX_RX – LL_SPI_HALF_DUPLEX_TX
Return values	• **None:**
Notes	• For Half-Duplex mode, Rx Direction is set by default. In master mode, the MOSI pin is used and in slave mode, the MISO pin is used for Half-Duplex.
Reference Manual to LL API cross reference:	• CR1 RXONLY LL_SPI_SetTransferDirection • CR1 BIDIMODE LL_SPI_SetTransferDirection • CR1 BIDIOE LL_SPI_SetTransferDirection

LL_SPI_GetTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetTransferDirection (SPI_TypeDef * SPIx)**
Function description	Get transfer direction mode.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_FULL_DUPLEX – LL_SPI_SIMPLEX_RX – LL_SPI_HALF_DUPLEX_RX – LL_SPI_HALF_DUPLEX_TX
Reference Manual to LL API cross reference:	• CR1 RXONLY LL_SPI_GetTransferDirection • CR1 BIDIMODE LL_SPI_GetTransferDirection • CR1 BIDIOE LL_SPI_GetTransferDirection

LL_SPI_SetDataWidth

Function name	**__STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)**
Function description	Set frame data width.

Parameters	• **SPIx:** SPI Instance • **DataWidth:** This parameter can be one of the following values: – LL_SPI_DATAWIDTH_4BIT – LL_SPI_DATAWIDTH_5BIT – LL_SPI_DATAWIDTH_6BIT – LL_SPI_DATAWIDTH_7BIT – LL_SPI_DATAWIDTH_8BIT – LL_SPI_DATAWIDTH_9BIT – LL_SPI_DATAWIDTH_10BIT – LL_SPI_DATAWIDTH_11BIT – LL_SPI_DATAWIDTH_12BIT – LL_SPI_DATAWIDTH_13BIT – LL_SPI_DATAWIDTH_14BIT – LL_SPI_DATAWIDTH_15BIT – LL_SPI_DATAWIDTH_16BIT
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 DS LL_SPI_SetDataWidth

LL_SPI_GetDataWidth

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetDataWidth (SPI_TypeDef * SPIx)**
Function description	Get frame data width.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_DATAWIDTH_4BIT – LL_SPI_DATAWIDTH_5BIT – LL_SPI_DATAWIDTH_6BIT – LL_SPI_DATAWIDTH_7BIT – LL_SPI_DATAWIDTH_8BIT – LL_SPI_DATAWIDTH_9BIT – LL_SPI_DATAWIDTH_10BIT – LL_SPI_DATAWIDTH_11BIT – LL_SPI_DATAWIDTH_12BIT – LL_SPI_DATAWIDTH_13BIT – LL_SPI_DATAWIDTH_14BIT – LL_SPI_DATAWIDTH_15BIT – LL_SPI_DATAWIDTH_16BIT
Reference Manual to LL API cross reference:	• CR2 DS LL_SPI_GetDataWidth

LL_SPI_SetRxFIFOThreshold

Function name	**__STATIC_INLINE void LL_SPI_SetRxFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)**

Function description	Set threshold of RXFIFO that triggers an RXNE event.
Parameters	• **SPIx:** SPI Instance • **Threshold:** This parameter can be one of the following values: – LL_SPI_RX_FIFO_TH_HALF – LL_SPI_RX_FIFO_TH_QUARTER
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 FRXTH LL_SPI_SetRxFIFOThreshold

LL_SPI_GetRxFIFOThreshold

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOThreshold (SPI_TypeDef * SPIx)**
Function description	Get threshold of RXFIFO that triggers an RXNE event.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_RX_FIFO_TH_HALF – LL_SPI_RX_FIFO_TH_QUARTER
Reference Manual to LL API cross reference:	• CR2 FRXTH LL_SPI_GetRxFIFOThreshold

LL_SPI_EnableCRC

Function name	**__STATIC_INLINE void LL_SPI_EnableCRC (SPI_TypeDef * SPIx)**
Function description	Enable CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	• CR1 CRCEN LL_SPI_EnableCRC

LL_SPI_DisableCRC

Function name	**__STATIC_INLINE void LL_SPI_DisableCRC (SPI_TypeDef * SPIx)**
Function description	Disable CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit should be written only when SPI is disabled (SPE = 0)

		for correct operation.
Reference Manual to LL API cross reference:	•	CR1 CRCEN LL_SPI_DisableCRC

LL_SPI_IsEnabledCRC

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledCRC (SPI_TypeDef * SPIx)**
Function description	Check if CRC is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Notes	• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	• CR1 CRCEN LL_SPI_IsEnabledCRC

LL_SPI_SetCRCWidth

Function name	**__STATIC_INLINE void LL_SPI_SetCRCWidth (SPI_TypeDef * SPIx, uint32_t CRCLength)**
Function description	Set CRC Length.
Parameters	• **SPIx:** SPI Instance • **CRCLength:** This parameter can be one of the following values: – LL_SPI_CRC_8BIT – LL_SPI_CRC_16BIT
Return values	• **None:**
Notes	• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	• CR1 CRCL LL_SPI_SetCRCWidth

LL_SPI_GetCRCWidth

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetCRCWidth (SPI_TypeDef * SPIx)**
Function description	Get CRC Length.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_CRC_8BIT – LL_SPI_CRC_16BIT
Reference Manual to LL API cross	• CR1 CRCL LL_SPI_GetCRCWidth

reference:

LL_SPI_SetCRCNext

Function name	**__STATIC_INLINE void LL_SPI_SetCRCNext (SPI_TypeDef * SPIx)**
Function description	Set CRCNext to transfer CRC on the line.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit has to be written as soon as the last data is written in the SPIx_DR register.
Reference Manual to LL API cross reference:	• CR1 CRCNEXT LL_SPI_SetCRCNext

LL_SPI_SetCRCPolynomial

Function name	**__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)**
Function description	Set polynomial for CRC calculation.
Parameters	• **SPIx:** SPI Instance • **CRCPoly:** This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• CRCPR CRCPOLY LL_SPI_SetCRCPolynomial

LL_SPI_GetCRCPolynomial

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetCRCPolynomial (SPI_TypeDef * SPIx)**
Function description	Get polynomial for CRC calculation.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• CRCPR CRCPOLY LL_SPI_GetCRCPolynomial

LL_SPI_GetRxCRC

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetRxCRC (SPI_TypeDef * SPIx)**
Function description	Get Rx CRC.
Parameters	• **SPIx:** SPI Instance

Return values	• **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• RXCRCR RXCRC LL_SPI_GetRxCRC

LL_SPI_GetTxCRC

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (SPI_TypeDef * SPIx)**
Function description	Get Tx CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• TXCRCR TXCRC LL_SPI_GetTxCRC

LL_SPI_SetNSSMode

Function name	**__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)**
Function description	Set NSS mode.
Parameters	• **SPIx:** SPI Instance • **NSS:** This parameter can be one of the following values: – LL_SPI_NSS_SOFT – LL_SPI_NSS_HARD_INPUT – LL_SPI_NSS_HARD_OUTPUT
Return values	• **None:**
Notes	• LL_SPI_NSS_SOFT Mode is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR1 SSM LL_SPI_SetNSSMode • • CR2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (SPI_TypeDef * SPIx)**
Function description	Get NSS mode.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_NSS_SOFT – LL_SPI_NSS_HARD_INPUT – LL_SPI_NSS_HARD_OUTPUT
Reference Manual to LL API cross	• CR1 SSM LL_SPI_GetNSSMode •

reference:	• CR2 SSOE LL_SPI_GetNSSMode

LL_SPI_EnableNSSPulseMgt

Function name	**__STATIC_INLINE void LL_SPI_EnableNSSPulseMgt (SPI_TypeDef * SPIx)**
Function description	Enable NSS pulse management.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR2 NSSP LL_SPI_EnableNSSPulseMgt

LL_SPI_DisableNSSPulseMgt

Function name	**__STATIC_INLINE void LL_SPI_DisableNSSPulseMgt (SPI_TypeDef * SPIx)**
Function description	Disable NSS pulse management.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR2 NSSP LL_SPI_DisableNSSPulseMgt

LL_SPI_IsEnabledNSSPulse

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledNSSPulse (SPI_TypeDef * SPIx)**
Function description	Check if NSS pulse is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Notes	• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR2 NSSP LL_SPI_IsEnabledNSSPulse

LL_SPI_IsActiveFlag_RXNE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXNE (SPI_TypeDef * SPIx)**

Function description	Check if Rx buffer is not empty.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR RXNE LL_SPI_IsActiveFlag_RXNE

LL_SPI_IsActiveFlag_TXE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXE (SPI_TypeDef * SPIx)**
Function description	Check if Tx buffer is empty.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR TXE LL_SPI_IsActiveFlag_TXE

LL_SPI_IsActiveFlag_CRCERR

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_CRCERR (SPI_TypeDef * SPIx)**
Function description	Get CRC error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CRCERR LL_SPI_IsActiveFlag_CRCERR

LL_SPI_IsActiveFlag_MODF

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_MODF (SPI_TypeDef * SPIx)**
Function description	Get mode fault error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR MODF LL_SPI_IsActiveFlag_MODF

LL_SPI_IsActiveFlag_OVR

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_OVR (SPI_TypeDef * SPIx)**
Function description	Get overrun error flag.

Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR OVR LL_SPI_IsActiveFlag_OVR

LL_SPI_IsActiveFlag_BSY

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_BSY (SPI_TypeDef * SPIx)**
Function description	Get busy flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Notes	• The BSY flag is cleared under any one of the following conditions: -When the SPI is correctly disabled -When a fault is detected in Master mode (MODF bit set to 1) -In Master mode, when it finishes a data transmission and no new data is ready to be sent -In Slave mode, when the BSY flag is set to '0' for at least one SPI clock cycle between each data transfer.
Reference Manual to LL API cross reference:	• SR BSY LL_SPI_IsActiveFlag_BSY

LL_SPI_IsActiveFlag_FRE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_FRE (SPI_TypeDef * SPIx)**
Function description	Get frame format error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR FRE LL_SPI_IsActiveFlag_FRE

LL_SPI_GetRxFIFOLevel

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOLevel (SPI_TypeDef * SPIx)**
Function description	Get FIFO reception Level.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_RX_FIFO_EMPTY – LL_SPI_RX_FIFO_QUARTER_FULL – LL_SPI_RX_FIFO_HALF_FULL – LL_SPI_RX_FIFO_FULL

Reference Manual to LL API cross reference:	• SR FRLVL LL_SPI_GetRxFIFOLevel

LL_SPI_GetTxFIFOLevel

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetTxFIFOLevel (SPI_TypeDef * SPIx)**
Function description	Get FIFO Transmission Level.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_TX_FIFO_EMPTY – LL_SPI_TX_FIFO_QUARTER_FULL – LL_SPI_TX_FIFO_HALF_FULL – LL_SPI_TX_FIFO_FULL
Reference Manual to LL API cross reference:	• SR FTLVL LL_SPI_GetTxFIFOLevel

LL_SPI_ClearFlag_CRCERR

Function name	**__STATIC_INLINE void LL_SPI_ClearFlag_CRCERR (SPI_TypeDef * SPIx)**
Function description	Clear CRC error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CRCERR LL_SPI_ClearFlag_CRCERR

LL_SPI_ClearFlag_MODF

Function name	**__STATIC_INLINE void LL_SPI_ClearFlag_MODF (SPI_TypeDef * SPIx)**
Function description	Clear mode fault error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the SPIx_SR register followed by a write access to the SPIx_CR1 register
Reference Manual to LL API cross reference:	• SR MODF LL_SPI_ClearFlag_MODF

LL_SPI_ClearFlag_OVR

Function name	**__STATIC_INLINE void LL_SPI_ClearFlag_OVR (SPI_TypeDef * SPIx)**

Function description	Clear overrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the SPIx_DR register followed by a read access to the SPIx_SR register
Reference Manual to LL API cross reference:	• SR OVR LL_SPI_ClearFlag_OVR

LL_SPI_ClearFlag_FRE

Function name	**__STATIC_INLINE void LL_SPI_ClearFlag_FRE (SPI_TypeDef * SPIx)**
Function description	Clear frame format error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• Clearing this flag is done by reading SPIx_SR register
Reference Manual to LL API cross reference:	• SR FRE LL_SPI_ClearFlag_FRE

LL_SPI_EnableIT_ERR

Function name	**__STATIC_INLINE void LL_SPI_EnableIT_ERR (SPI_TypeDef * SPIx)**
Function description	Enable error interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_SPI_EnableIT_ERR

LL_SPI_EnableIT_RXNE

Function name	**__STATIC_INLINE void LL_SPI_EnableIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Enable Rx buffer not empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross	• CR2 RXNEIE LL_SPI_EnableIT_RXNE

reference:

LL_SPI_EnableIT_TXE

Function name	**__STATIC_INLINE void LL_SPI_EnableIT_TXE (SPI_TypeDef * SPIx)**
Function description	Enable Tx buffer empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_SPI_EnableIT_TXE

LL_SPI_DisableIT_ERR

Function name	**__STATIC_INLINE void LL_SPI_DisableIT_ERR (SPI_TypeDef * SPIx)**
Function description	Disable error interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_SPI_DisableIT_ERR

LL_SPI_DisableIT_RXNE

Function name	**__STATIC_INLINE void LL_SPI_DisableIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Disable Rx buffer not empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_SPI_DisableIT_RXNE

LL_SPI_DisableIT_TXE

Function name	**__STATIC_INLINE void LL_SPI_DisableIT_TXE (SPI_TypeDef * SPIx)**
Function description	Disable Tx buffer empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• CR2 TXEIE LL_SPI_DisableIT_TXE

LL_SPI_IsEnabledIT_ERR

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_ERR (SPI_TypeDef * SPIx)**
Function description	Check if error interrupt is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_SPI_IsEnabledIT_ERR

LL_SPI_IsEnabledIT_RXNE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Check if Rx buffer not empty interrupt is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_SPI_IsEnabledIT_RXNE

LL_SPI_IsEnabledIT_TXE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXE (SPI_TypeDef * SPIx)**
Function description	Check if Tx buffer empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_SPI_IsEnabledIT_TXE

LL_SPI_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_SPI_EnableDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Enable DMA Rx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross	• CR2 RXDMAEN LL_SPI_EnableDMAReq_RX

reference:	

LL_SPI_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_SPI_DisableDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Disable DMA Rx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_SPI_DisableDMAReq_RX

LL_SPI_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Check if DMA Rx is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_SPI_IsEnabledDMAReq_RX

LL_SPI_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_SPI_EnableDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Enable DMA Tx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_SPI_EnableDMAReq_TX

LL_SPI_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_SPI_DisableDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Disable DMA Tx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_SPI_DisableDMAReq_TX

LL_SPI_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Check if DMA Tx is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_SPI_IsEnabledDMAReq_TX

LL_SPI_SetDMAParity_RX

Function name	**__STATIC_INLINE void LL_SPI_SetDMAParity_RX (SPI_TypeDef * SPIx, uint32_t Parity)**
Function description	Set parity of Last DMA reception.
Parameters	• **SPIx:** SPI Instance • **Parity:** This parameter can be one of the following values: – LL_SPI_DMA_PARITY_ODD – LL_SPI_DMA_PARITY_EVEN
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 LDMARX LL_SPI_SetDMAParity_RX

LL_SPI_GetDMAParity_RX

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetDMAParity_RX (SPI_TypeDef * SPIx)**
Function description	Get parity configuration for Last DMA reception.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_DMA_PARITY_ODD – LL_SPI_DMA_PARITY_EVEN
Reference Manual to LL API cross reference:	• CR2 LDMARX LL_SPI_GetDMAParity_RX

LL_SPI_SetDMAParity_TX

Function name	**__STATIC_INLINE void LL_SPI_SetDMAParity_TX (SPI_TypeDef * SPIx, uint32_t Parity)**
Function description	Set parity of Last DMA transmission.
Parameters	• **SPIx:** SPI Instance • **Parity:** This parameter can be one of the following values: – LL_SPI_DMA_PARITY_ODD – LL_SPI_DMA_PARITY_EVEN

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 LDMATX LL_SPI_SetDMAParity_TX

LL_SPI_GetDMAParity_TX

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetDMAParity_TX (SPI_TypeDef * SPIx)**
Function description	Get parity configuration for Last DMA transmission.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_DMA_PARITY_ODD – LL_SPI_DMA_PARITY_EVEN
Reference Manual to LL API cross reference:	• CR2 LDMATX LL_SPI_GetDMAParity_TX

LL_SPI_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_SPI_DMA_GetRegAddr (SPI_TypeDef * SPIx)**
Function description	Get the data register address used for DMA transfer.
Parameters	• **SPIx:** SPI Instance
Return values	• **Address:** of data register
Reference Manual to LL API cross reference:	• DR DR LL_SPI_DMA_GetRegAddr

LL_SPI_ReceiveData8

Function name	**__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)**
Function description	Read 8-Bits in the data register.
Parameters	• **SPIx:** SPI Instance
Return values	• **RxData:** Value between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• DR DR LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name	**__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)**
Function description	Read 16-Bits in the data register.

Parameters	• **SPIx:** SPI Instance
Return values	• **RxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• DR DR LL_SPI_ReceiveData16

LL_SPI_TransmitData8

Function name	**__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)**
Function description	Write 8-Bits in the data register.
Parameters	• **SPIx:** SPI Instance • **TxData:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_SPI_TransmitData8

LL_SPI_TransmitData16

Function name	**__STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)**
Function description	Write 16-Bits in the data register.
Parameters	• **SPIx:** SPI Instance • **TxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_SPI_TransmitData16

LL_SPI_DeInit

Function name	**ErrorStatus LL_SPI_DeInit (SPI_TypeDef * SPIx)**
Function description	De-initialize the SPI registers to their default reset values.
Parameters	• **SPIx:** SPI Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: SPI registers are de-initialized – ERROR: SPI registers are not de-initialized

LL_SPI_Init

Function name	**ErrorStatus LL_SPI_Init (SPI_TypeDef * SPIx, LL_SPI_InitTypeDef * SPI_InitStruct)**

Function description	Initialize the SPI registers according to the specified parameters in SPI_InitStruct.
Parameters	• **SPIx:** SPI Instance • **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value. (Return always SUCCESS)
Notes	• As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_SPI_StructInit

Function name	**void LL_SPI_StructInit (LL_SPI_InitTypeDef * SPI_InitStruct)**
Function description	Set each LL_SPI_InitTypeDef field to default value.
Parameters	• **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

94.3 SPI Firmware driver defines

94.3.1 SPI

Baud Rate Prescaler

LL_SPI_BAUDRATEPRESCALER_DIV2	BaudRate control equal to fPCLK/2
LL_SPI_BAUDRATEPRESCALER_DIV4	BaudRate control equal to fPCLK/4
LL_SPI_BAUDRATEPRESCALER_DIV8	BaudRate control equal to fPCLK/8
LL_SPI_BAUDRATEPRESCALER_DIV16	BaudRate control equal to fPCLK/16
LL_SPI_BAUDRATEPRESCALER_DIV32	BaudRate control equal to fPCLK/32
LL_SPI_BAUDRATEPRESCALER_DIV64	BaudRate control equal to fPCLK/64
LL_SPI_BAUDRATEPRESCALER_DIV128	BaudRate control equal to fPCLK/128
LL_SPI_BAUDRATEPRESCALER_DIV256	BaudRate control equal to fPCLK/256

Transmission Bit Order

LL_SPI_LSB_FIRST	Data is transmitted/received with the LSB first
LL_SPI_MSB_FIRST	Data is transmitted/received with the MSB first

CRC Calculation

LL_SPI_CRCCALCULATION_DISABLE	CRC calculation disabled
LL_SPI_CRCCALCULATION_ENABLE	CRC calculation enabled

CRC Length

LL_SPI_CRC_8BIT	8-bit CRC length
LL_SPI_CRC_16BIT	16-bit CRC length

Datawidth

LL_SPI_DATAWIDTH_4BIT	Data length for SPI transfer: 4 bits
LL_SPI_DATAWIDTH_5BIT	Data length for SPI transfer: 5 bits
LL_SPI_DATAWIDTH_6BIT	Data length for SPI transfer: 6 bits
LL_SPI_DATAWIDTH_7BIT	Data length for SPI transfer: 7 bits
LL_SPI_DATAWIDTH_8BIT	Data length for SPI transfer: 8 bits
LL_SPI_DATAWIDTH_9BIT	Data length for SPI transfer: 9 bits
LL_SPI_DATAWIDTH_10BIT	Data length for SPI transfer: 10 bits
LL_SPI_DATAWIDTH_11BIT	Data length for SPI transfer: 11 bits
LL_SPI_DATAWIDTH_12BIT	Data length for SPI transfer: 12 bits
LL_SPI_DATAWIDTH_13BIT	Data length for SPI transfer: 13 bits
LL_SPI_DATAWIDTH_14BIT	Data length for SPI transfer: 14 bits
LL_SPI_DATAWIDTH_15BIT	Data length for SPI transfer: 15 bits
LL_SPI_DATAWIDTH_16BIT	Data length for SPI transfer: 16 bits

DMA Parity

LL_SPI_DMA_PARITY_EVEN	Select DMA parity Even
LL_SPI_DMA_PARITY_ODD	Select DMA parity Odd

Get Flags Defines

LL_SPI_SR_RXNE	Rx buffer not empty flag
LL_SPI_SR_TXE	Tx buffer empty flag
LL_SPI_SR_BSY	Busy flag
LL_SPI_SR_CRCERR	CRC error flag
LL_SPI_SR_MODF	Mode fault flag
LL_SPI_SR_OVR	Overrun flag
LL_SPI_SR_FRE	TI mode frame format error flag

IT Defines

LL_SPI_CR2_RXNEIE	Rx buffer not empty interrupt enable
LL_SPI_CR2_TXEIE	Tx buffer empty interrupt enable
LL_SPI_CR2_ERRIE	Error interrupt enable

Operation Mode

LL_SPI_MODE_MASTER	Master configuration
LL_SPI_MODE_SLAVE	Slave configuration

Slave Select Pin Mode

LL_SPI_NSS_SOFT	NSS managed internally. NSS pin not used and free
LL_SPI_NSS_HARD_INPUT	NSS pin used in Input. Only used in Master mode
LL_SPI_NSS_HARD_OUTPUT	NSS pin used in Output. Only used in Slave mode as chip select

Clock Phase

LL_SPI_PHASE_1EDGE	First clock transition is the first data capture edge
LL_SPI_PHASE_2EDGE	Second clock transition is the first data capture edge

Clock Polarity

LL_SPI_POLARITY_LOW	Clock to 0 when idle
LL_SPI_POLARITY_HIGH	Clock to 1 when idle

Serial Protocol

LL_SPI_PROTOCOL_MOTOROLA	Motorola mode. Used as default value
LL_SPI_PROTOCOL_TI	TI mode

RX FIFO Level

LL_SPI_RX_FIFO_EMPTY	FIFO reception empty
LL_SPI_RX_FIFO_QUARTER_FULL	FIFO reception 1/4
LL_SPI_RX_FIFO_HALF_FULL	FIFO reception 1/2
LL_SPI_RX_FIFO_FULL	FIFO reception full

RX FIFO Threshold

LL_SPI_RX_FIFO_TH_HALF	RXNE event is generated if FIFO level is greater than or equel to 1/2 (16-bit)
LL_SPI_RX_FIFO_TH_QUARTER	RXNE event is generated if FIFO level is greater than or equel to 1/4 (8-bit)

Transfer Mode

LL_SPI_FULL_DUPLEX	Full-Duplex mode. Rx and Tx transfer on 2 lines
LL_SPI_SIMPLEX_RX	Simplex Rx mode. Rx transfer only on 1 line
LL_SPI_HALF_DUPLEX_RX	Half-Duplex Rx mode. Rx transfer on 1 line
LL_SPI_HALF_DUPLEX_TX	Half-Duplex Tx mode. Tx transfer on 1 line

TX FIFO Level

LL_SPI_TX_FIFO_EMPTY	FIFO transmission empty
LL_SPI_TX_FIFO_QUARTER_FULL	FIFO transmission 1/4
LL_SPI_TX_FIFO_HALF_FULL	FIFO transmission 1/2
LL_SPI_TX_FIFO_FULL	FIFO transmission full

Common Write and read registers Macros

LL_SPI_WriteReg

Description:

- Write a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_SPI_ReadReg

Description:

- Read a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be read

Return value:

- Register: value

95 LL SYSTEM Generic Driver

95.1 SYSTEM Firmware driver API description

95.1.1 Detailed description of functions

LL_SYSCFG_SetRemapMemory

Function name	**__STATIC_INLINE void LL_SYSCFG_SetRemapMemory (uint32_t Memory)**
Function description	Set memory mapping at address 0x00000000.
Parameters	• **Memory:** This parameter can be one of the following values: (*) value not defined in all devices – LL_SYSCFG_REMAP_FLASH – LL_SYSCFG_REMAP_SYSTEMFLASH – LL_SYSCFG_REMAP_SRAM – LL_SYSCFG_REMAP_FMC (*) – LL_SYSCFG_REMAP_QUADSPI
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_MEMRMP MEM_MODE LL_SYSCFG_SetRemapMemory

LL_SYSCFG_GetRemapMemory

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_GetRemapMemory (void)**
Function description	Get memory mapping at address 0x00000000.
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices – LL_SYSCFG_REMAP_FLASH – LL_SYSCFG_REMAP_SYSTEMFLASH – LL_SYSCFG_REMAP_SRAM – LL_SYSCFG_REMAP_FMC (*) – LL_SYSCFG_REMAP_QUADSPI
Reference Manual to LL API cross reference:	• SYSCFG_MEMRMP MEM_MODE LL_SYSCFG_GetRemapMemory

LL_SYSCFG_SetFlashBankMode

Function name	**__STATIC_INLINE void LL_SYSCFG_SetFlashBankMode (uint32_t Bank)**
Function description	Select Flash bank mode (Bank flashed at 0x08000000)
Parameters	• **Bank:** This parameter can be one of the following values: – LL_SYSCFG_BANKMODE_BANK1

– LL_SYSCFG_BANKMODE_BANK2

Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_MEMRMP FB_MODE LL_SYSCFG_SetFlashBankMode

LL_SYSCFG_GetFlashBankMode

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashBankMode (void)**
Function description	Get Flash bank mode (Bank flashed at 0x08000000)
Return values	• **Returned:** value can be one of the following values: – LL_SYSCFG_BANKMODE_BANK1 – LL_SYSCFG_BANKMODE_BANK2
Reference Manual to LL API cross reference:	• SYSCFG_MEMRMP FB_MODE LL_SYSCFG_GetFlashBankMode

LL_SYSCFG_EnableFirewall

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableFirewall (void)**
Function description	Firewall protection enabled.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FWDIS LL_SYSCFG_EnableFirewall

LL_SYSCFG_IsEnabledFirewall

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledFirewall (void)**
Function description	Check if Firewall protection is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FWDIS LL_SYSCFG_IsEnabledFirewall

LL_SYSCFG_EnableAnalogBooster

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableAnalogBooster (void)**
Function description	Enable I/O analog switch voltage booster.
Return values	• **None:**
Notes	• When voltage booster is enabled, I/O analog switches are supplied by a dedicated voltage booster, from VDD power domain. This is the recommended configuration with low VDDA voltage operation.

- The I/O analog switch voltage booster is relevant for peripherals using I/O in analog input: ADC, COMP, OPAMP. However, COMP and OPAMP inputs have a high impedance and voltage booster do not impact performance significantly. Therefore, the voltage booster is mainly intended for usage with ADC.

Reference Manual to LL API cross reference:
- SYSCFG_CFGR1 BOOSTEN LL_SYSCFG_EnableAnalogBooster

LL_SYSCFG_DisableAnalogBooster

Function name
__STATIC_INLINE void LL_SYSCFG_DisableAnalogBooster (void)

Function description
Disable I/O analog switch voltage booster.

Return values
- **None:**

Notes
- When voltage booster is enabled, I/O analog switches are supplied by a dedicated voltage booster, from VDD power domain. This is the recommended configuration with low VDDA voltage operation.
- The I/O analog switch voltage booster is relevant for peripherals using I/O in analog input: ADC, COMP, OPAMP. However, COMP and OPAMP inputs have a high impedance and voltage booster do not impact performance significantly. Therefore, the voltage booster is mainly intended for usage with ADC.

Reference Manual to LL API cross reference:
- SYSCFG_CFGR1 BOOSTEN LL_SYSCFG_DisableAnalogBooster

LL_SYSCFG_EnableFastModePlus

Function name
__STATIC_INLINE void LL_SYSCFG_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description
Enable the I2C fast mode plus driving capability.

Parameters
- **ConfigFastModePlus:** This parameter can be a combination of the following values: (*) value not defined in all devices
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB7
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB8 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB9 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C1
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C2 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C3
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C4 (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- SYSCFG_CFGR1 I2C_PBx_FMP LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2Cx_FMP

LL_SYSCFG_EnableFastModePlus

LL_SYSCFG_DisableFastModePlus

Function name	**__STATIC_INLINE void LL_SYSCFG_DisableFastModePlus (uint32_t ConfigFastModePlus)**
Function description	Disable the I2C fast mode plus driving capability.
Parameters	• **ConfigFastModePlus:** This parameter can be a combination of the following values: (*) value not defined in all devices – LL_SYSCFG_I2C_FASTMODEPLUS_PB6 – LL_SYSCFG_I2C_FASTMODEPLUS_PB7 – LL_SYSCFG_I2C_FASTMODEPLUS_PB8 (*) – LL_SYSCFG_I2C_FASTMODEPLUS_PB9 (*) – LL_SYSCFG_I2C_FASTMODEPLUS_I2C1 – LL_SYSCFG_I2C_FASTMODEPLUS_I2C2 (*) – LL_SYSCFG_I2C_FASTMODEPLUS_I2C3 – LL_SYSCFG_I2C_FASTMODEPLUS_I2C4 (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 I2C_PBx_FMP LL_SYSCFG_DisableFastModePlus • SYSCFG_CFGR1 I2Cx_FMP LL_SYSCFG_DisableFastModePlus

LL_SYSCFG_EnableIT_FPU_IOC

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_IOC (void)**
Function description	Enable Floating Point Unit Invalid operation Interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_0 LL_SYSCFG_EnableIT_FPU_IOC

LL_SYSCFG_EnableIT_FPU_DZC

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_DZC (void)**
Function description	Enable Floating Point Unit Divide-by-zero Interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_1 LL_SYSCFG_EnableIT_FPU_DZC

LL_SYSCFG_EnableIT_FPU_UFC

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_UFC (void)**
Function description	Enable Floating Point Unit Underflow Interrupt.

Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_2 LL_SYSCFG_EnableIT_FPU_UFC

LL_SYSCFG_EnableIT_FPU_OFC

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_OFC (void)**
Function description	Enable Floating Point Unit Overflow Interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_3 LL_SYSCFG_EnableIT_FPU_OFC

LL_SYSCFG_EnableIT_FPU_IDC

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_IDC (void)**
Function description	Enable Floating Point Unit Input denormal Interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_4 LL_SYSCFG_EnableIT_FPU_IDC

LL_SYSCFG_EnableIT_FPU_IXC

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_IXC (void)**
Function description	Enable Floating Point Unit Inexact Interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_5 LL_SYSCFG_EnableIT_FPU_IXC

LL_SYSCFG_DisableIT_FPU_IOC

Function name	**__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_IOC (void)**
Function description	Disable Floating Point Unit Invalid operation Interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_0 LL_SYSCFG_DisableIT_FPU_IOC

LL_SYSCFG_DisableIT_FPU_DZC

Function name	**__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_DZC (void)**
Function description	Disable Floating Point Unit Divide-by-zero Interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_1 LL_SYSCFG_DisableIT_FPU_DZC

LL_SYSCFG_DisableIT_FPU_UFC

Function name	**__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_UFC (void)**
Function description	Disable Floating Point Unit Underflow Interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_2 LL_SYSCFG_DisableIT_FPU_UFC

LL_SYSCFG_DisableIT_FPU_OFC

Function name	**__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_OFC (void)**
Function description	Disable Floating Point Unit Overflow Interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_3 LL_SYSCFG_DisableIT_FPU_OFC

LL_SYSCFG_DisableIT_FPU_IDC

Function name	**__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_IDC (void)**
Function description	Disable Floating Point Unit Input denormal Interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_4 LL_SYSCFG_DisableIT_FPU_IDC

LL_SYSCFG_DisableIT_FPU_IXC

Function name	**__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_IXC (void)**
Function description	Disable Floating Point Unit Inexact Interrupt.
Return values	• **None:**
Reference Manual to	• SYSCFG_CFGR1 FPU_IE_5

LL API cross reference:	LL_SYSCFG_DisableIT_FPU_IXC

LL_SYSCFG_IsEnabledIT_FPU_IOC

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_IOC (void)**
Function description	Check if Floating Point Unit Invalid operation Interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_0 LL_SYSCFG_IsEnabledIT_FPU_IOC

LL_SYSCFG_IsEnabledIT_FPU_DZC

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_DZC (void)**
Function description	Check if Floating Point Unit Divide-by-zero Interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_1 LL_SYSCFG_IsEnabledIT_FPU_DZC

LL_SYSCFG_IsEnabledIT_FPU_UFC

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_UFC (void)**
Function description	Check if Floating Point Unit Underflow Interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_2 LL_SYSCFG_IsEnabledIT_FPU_UFC

LL_SYSCFG_IsEnabledIT_FPU_OFC

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_OFC (void)**
Function description	Check if Floating Point Unit Overflow Interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_3 LL_SYSCFG_IsEnabledIT_FPU_OFC

LL_SYSCFG_IsEnabledIT_FPU_IDC

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_IDC (void)**
Function description	Check if Floating Point Unit Input denormal Interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_4 LL_SYSCFG_IsEnabledIT_FPU_IDC

LL_SYSCFG_IsEnabledIT_FPU_IXC

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_IXC (void)**
Function description	Check if Floating Point Unit Inexact Interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SYSCFG_CFGR1 FPU_IE_5 LL_SYSCFG_IsEnabledIT_FPU_IXC

LL_SYSCFG_SetEXTISource

Function name	**__STATIC_INLINE void LL_SYSCFG_SetEXTISource (uint32_t Port, uint32_t Line)**
Function description	Configure source input for the EXTI external interrupt.
Parameters	• **Port:** This parameter can be one of the following values: (*) value not defined in all devices – LL_SYSCFG_EXTI_PORTA – LL_SYSCFG_EXTI_PORTB – LL_SYSCFG_EXTI_PORTC – LL_SYSCFG_EXTI_PORTD – LL_SYSCFG_EXTI_PORTE – LL_SYSCFG_EXTI_PORTF (*) – LL_SYSCFG_EXTI_PORTG (*) – LL_SYSCFG_EXTI_PORTH – LL_SYSCFG_EXTI_PORTI (*) • **Line:** This parameter can be one of the following values: – LL_SYSCFG_EXTI_LINE0 – LL_SYSCFG_EXTI_LINE1 – LL_SYSCFG_EXTI_LINE2 – LL_SYSCFG_EXTI_LINE3 – LL_SYSCFG_EXTI_LINE4 – LL_SYSCFG_EXTI_LINE5 – LL_SYSCFG_EXTI_LINE6 – LL_SYSCFG_EXTI_LINE7 – LL_SYSCFG_EXTI_LINE8 – LL_SYSCFG_EXTI_LINE9 – LL_SYSCFG_EXTI_LINE10

- – LL_SYSCFG_EXTI_LINE11
- – LL_SYSCFG_EXTI_LINE12
- – LL_SYSCFG_EXTI_LINE13
- – LL_SYSCFG_EXTI_LINE14
- – LL_SYSCFG_EXTI_LINE15

Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_EXTICR1 EXTIx LL_SYSCFG_SetEXTISource • SYSCFG_EXTICR2 EXTIx LL_SYSCFG_SetEXTISource • SYSCFG_EXTICR3 EXTIx LL_SYSCFG_SetEXTISource • SYSCFG_EXTICR4 EXTIx LL_SYSCFG_SetEXTISource

LL_SYSCFG_GetEXTISource

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_GetEXTISource (uint32_t Line)**
Function description	Get the configured defined for specific EXTI Line.
Parameters	• **Line:** This parameter can be one of the following values: – LL_SYSCFG_EXTI_LINE0 – LL_SYSCFG_EXTI_LINE1 – LL_SYSCFG_EXTI_LINE2 – LL_SYSCFG_EXTI_LINE3 – LL_SYSCFG_EXTI_LINE4 – LL_SYSCFG_EXTI_LINE5 – LL_SYSCFG_EXTI_LINE6 – LL_SYSCFG_EXTI_LINE7 – LL_SYSCFG_EXTI_LINE8 – LL_SYSCFG_EXTI_LINE9 – LL_SYSCFG_EXTI_LINE10 – LL_SYSCFG_EXTI_LINE11 – LL_SYSCFG_EXTI_LINE12 – LL_SYSCFG_EXTI_LINE13 – LL_SYSCFG_EXTI_LINE14 – LL_SYSCFG_EXTI_LINE15
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices – LL_SYSCFG_EXTI_PORTA – LL_SYSCFG_EXTI_PORTB – LL_SYSCFG_EXTI_PORTC – LL_SYSCFG_EXTI_PORTD – LL_SYSCFG_EXTI_PORTE – LL_SYSCFG_EXTI_PORTF (*) – LL_SYSCFG_EXTI_PORTG (*) – LL_SYSCFG_EXTI_PORTH – LL_SYSCFG_EXTI_PORTI (*)
Reference Manual to LL API cross reference:	• SYSCFG_EXTICR1 EXTIx LL_SYSCFG_GetEXTISource • SYSCFG_EXTICR2 EXTIx LL_SYSCFG_GetEXTISource • SYSCFG_EXTICR3 EXTIx LL_SYSCFG_GetEXTISource • SYSCFG_EXTICR4 EXTIx LL_SYSCFG_GetEXTISource

LL_SYSCFG_EnableSRAM2Erase

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableSRAM2Erase (void)**
Function description	Enable SRAM2 Erase (starts a hardware SRAM2 erase operation.
Return values	• **None:**
Notes	• This bit is write-protected: setting this bit is possible only after the correct key sequence is written in the SYSCFG_SKR register as described in the Reference Manual.
Reference Manual to LL API cross reference:	• SYSCFG_SCSR SRAM2ER LL_SYSCFG_EnableSRAM2Erase

LL_SYSCFG_IsSRAM2EraseOngoing

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_IsSRAM2EraseOngoing (void)**
Function description	Check if SRAM2 erase operation is on going.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SYSCFG_SCSR SRAM2BSY LL_SYSCFG_IsSRAM2EraseOngoing

LL_SYSCFG_SetTIMBreakInputs

Function name	**__STATIC_INLINE void LL_SYSCFG_SetTIMBreakInputs (uint32_t Break)**
Function description	Set connections to TIM1/8/15/16/17 Break inputs.
Parameters	• **Break:** This parameter can be a combination of the following values: – LL_SYSCFG_TIMBREAK_ECC – LL_SYSCFG_TIMBREAK_PVD – LL_SYSCFG_TIMBREAK_SRAM2_PARITY – LL_SYSCFG_TIMBREAK_LOCKUP
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR2 CLL LL_SYSCFG_SetTIMBreakInputs • SYSCFG_CFGR2 SPL LL_SYSCFG_SetTIMBreakInputs • SYSCFG_CFGR2 PVDL LL_SYSCFG_SetTIMBreakInputs • SYSCFG_CFGR2 ECCL LL_SYSCFG_SetTIMBreakInputs

LL_SYSCFG_GetTIMBreakInputs

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_GetTIMBreakInputs (void)**
Function description	Get connections to TIM1/8/15/16/17 Break inputs.
Return values	• **Returned:** value can be can be a combination of the following values: – LL_SYSCFG_TIMBREAK_ECC

- LL_SYSCFG_TIMBREAK_PVD
- LL_SYSCFG_TIMBREAK_SRAM2_PARITY
- LL_SYSCFG_TIMBREAK_LOCKUP

Reference Manual to LL API cross reference:	• SYSCFG_CFGR2 CLL LL_SYSCFG_GetTIMBreakInputs • SYSCFG_CFGR2 SPL LL_SYSCFG_GetTIMBreakInputs • SYSCFG_CFGR2 PVDL LL_SYSCFG_GetTIMBreakInputs • SYSCFG_CFGR2 ECCL LL_SYSCFG_GetTIMBreakInputs

LL_SYSCFG_IsActiveFlag_SP

Function name	**__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_SP (void)**
Function description	Check if SRAM2 parity error detected.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SYSCFG_CFGR2 SPF LL_SYSCFG_IsActiveFlag_SP

LL_SYSCFG_ClearFlag_SP

Function name	**__STATIC_INLINE void LL_SYSCFG_ClearFlag_SP (void)**
Function description	Clear SRAM2 parity error flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_CFGR2 SPF LL_SYSCFG_ClearFlag_SP

LL_SYSCFG_EnableSRAM2PageWRP_0_31

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableSRAM2PageWRP_0_31 (uint32_t SRAM2WRP)**
Function description	

LL_SYSCFG_EnableSRAM2PageWRP_32_63

Function name	**__STATIC_INLINE void LL_SYSCFG_EnableSRAM2PageWRP_32_63 (uint32_t SRAM2WRP)**
Function description	Enable SRAM2 page write protection for Pages in range 32 to 63.
Parameters	• **SRAM2WRP:** This parameter can be a combination of the following values: (*) value not defined in all devices – LL_SYSCFG_SRAM2WRP_PAGE32 (*) – LL_SYSCFG_SRAM2WRP_PAGE33 (*) – LL_SYSCFG_SRAM2WRP_PAGE34 (*) – LL_SYSCFG_SRAM2WRP_PAGE35 (*) – LL_SYSCFG_SRAM2WRP_PAGE36 (*) – LL_SYSCFG_SRAM2WRP_PAGE37 (*) – LL_SYSCFG_SRAM2WRP_PAGE38 (*)

- LL_SYSCFG_SRAM2WRP_PAGE39 (*)
- LL_SYSCFG_SRAM2WRP_PAGE40 (*)
- LL_SYSCFG_SRAM2WRP_PAGE41 (*)
- LL_SYSCFG_SRAM2WRP_PAGE42 (*)
- LL_SYSCFG_SRAM2WRP_PAGE43 (*)
- LL_SYSCFG_SRAM2WRP_PAGE44 (*)
- LL_SYSCFG_SRAM2WRP_PAGE45 (*)
- LL_SYSCFG_SRAM2WRP_PAGE46 (*)
- LL_SYSCFG_SRAM2WRP_PAGE47 (*)
- LL_SYSCFG_SRAM2WRP_PAGE48 (*)
- LL_SYSCFG_SRAM2WRP_PAGE49 (*)
- LL_SYSCFG_SRAM2WRP_PAGE50 (*)
- LL_SYSCFG_SRAM2WRP_PAGE51 (*)
- LL_SYSCFG_SRAM2WRP_PAGE52 (*)
- LL_SYSCFG_SRAM2WRP_PAGE53 (*)
- LL_SYSCFG_SRAM2WRP_PAGE54 (*)
- LL_SYSCFG_SRAM2WRP_PAGE55 (*)
- LL_SYSCFG_SRAM2WRP_PAGE56 (*)
- LL_SYSCFG_SRAM2WRP_PAGE57 (*)
- LL_SYSCFG_SRAM2WRP_PAGE58 (*)
- LL_SYSCFG_SRAM2WRP_PAGE59 (*)
- LL_SYSCFG_SRAM2WRP_PAGE60 (*)
- LL_SYSCFG_SRAM2WRP_PAGE61 (*)
- LL_SYSCFG_SRAM2WRP_PAGE62 (*)
- LL_SYSCFG_SRAM2WRP_PAGE63 (*)

Return values	• **None:**
Notes	• Write protection is cleared only by a system reset
Reference Manual to LL API cross reference:	• SYSCFG_SWPR2 PxWP LL_SYSCFG_EnableSRAM2PageWRP_32_63

LL_SYSCFG_LockSRAM2WRP

Function name	**__STATIC_INLINE void LL_SYSCFG_LockSRAM2WRP (void)**
Function description	SRAM2 page write protection lock prior to erase.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SYSCFG_SKR KEY LL_SYSCFG_LockSRAM2WRP

LL_SYSCFG_UnlockSRAM2WRP

Function name	**__STATIC_INLINE void LL_SYSCFG_UnlockSRAM2WRP (void)**
Function description	SRAM2 page write protection unlock prior to erase.
Return values	• **None:**
Reference Manual to LL API cross	• SYSCFG_SKR KEY LL_SYSCFG_UnlockSRAM2WRP

reference:

LL_DBGMCU_GetDeviceID

Function name	**__STATIC_INLINE uint32_t LL_DBGMCU_GetDeviceID (void)**
Function description	Return the device identifier.
Return values	• **Values:** between Min_Data=0x00 and Max_Data=0xFFFF (ex: device ID is 0x6415)
Reference Manual to LL API cross reference:	• DBGMCU_IDCODE DEV_ID LL_DBGMCU_GetDeviceID

LL_DBGMCU_GetRevisionID

Function name	**__STATIC_INLINE uint32_t LL_DBGMCU_GetRevisionID (void)**
Function description	Return the device revision identifier.
Return values	• **Values:** between Min_Data=0x00 and Max_Data=0xFFFF
Notes	• This field indicates the revision of the device.
Reference Manual to LL API cross reference:	• DBGMCU_IDCODE REV_ID LL_DBGMCU_GetRevisionID

LL_DBGMCU_EnableDBGSleepMode

Function name	**__STATIC_INLINE void LL_DBGMCU_EnableDBGSleepMode (void)**
Function description	Enable the Debug Module during SLEEP mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_SLEEP LL_DBGMCU_EnableDBGSleepMode

LL_DBGMCU_DisableDBGSleepMode

Function name	**__STATIC_INLINE void LL_DBGMCU_DisableDBGSleepMode (void)**
Function description	Disable the Debug Module during SLEEP mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_SLEEP LL_DBGMCU_DisableDBGSleepMode

LL_DBGMCU_EnableDBGStopMode

Function name	**__STATIC_INLINE void LL_DBGMCU_EnableDBGStopMode (void)**

Function description	Enable the Debug Module during STOP mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_STOP LL_DBGMCU_EnableDBGStopMode

LL_DBGMCU_DisableDBGStopMode

Function name	**__STATIC_INLINE void LL_DBGMCU_DisableDBGStopMode (void)**
Function description	Disable the Debug Module during STOP mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_STOP LL_DBGMCU_DisableDBGStopMode

LL_DBGMCU_EnableDBGStandbyMode

Function name	**__STATIC_INLINE void LL_DBGMCU_EnableDBGStandbyMode (void)**
Function description	Enable the Debug Module during STANDBY mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_STANDBY LL_DBGMCU_EnableDBGStandbyMode

LL_DBGMCU_DisableDBGStandbyMode

Function name	**__STATIC_INLINE void LL_DBGMCU_DisableDBGStandbyMode (void)**
Function description	Disable the Debug Module during STANDBY mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_STANDBY LL_DBGMCU_DisableDBGStandbyMode

LL_DBGMCU_SetTracePinAssignment

Function name	**__STATIC_INLINE void LL_DBGMCU_SetTracePinAssignment (uint32_t PinAssignment)**
Function description	Set Trace pin assignment control.
Parameters	• **PinAssignment:** This parameter can be one of the following values: – LL_DBGMCU_TRACE_NONE – LL_DBGMCU_TRACE_ASYNCH – LL_DBGMCU_TRACE_SYNCH_SIZE1 – LL_DBGMCU_TRACE_SYNCH_SIZE2

– LL_DBGMCU_TRACE_SYNCH_SIZE4

Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_CR TRACE_IOEN LL_DBGMCU_SetTracePinAssignment • DBGMCU_CR TRACE_MODE LL_DBGMCU_SetTracePinAssignment

LL_DBGMCU_GetTracePinAssignment

Function name	**__STATIC_INLINE uint32_t LL_DBGMCU_GetTracePinAssignment (void)**
Function description	Get Trace pin assignment control.
Return values	• **Returned:** value can be one of the following values: – LL_DBGMCU_TRACE_NONE – LL_DBGMCU_TRACE_ASYNCH – LL_DBGMCU_TRACE_SYNCH_SIZE1 – LL_DBGMCU_TRACE_SYNCH_SIZE2 – LL_DBGMCU_TRACE_SYNCH_SIZE4
Reference Manual to LL API cross reference:	• DBGMCU_CR TRACE_IOEN LL_DBGMCU_GetTracePinAssignment • DBGMCU_CR TRACE_MODE LL_DBGMCU_GetTracePinAssignment

LL_DBGMCU_APB1_GRP1_FreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_FreezePeriph (uint32_t Periphs)**
Function description	Freeze APB1 peripherals (group1 peripherals)
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_DBGMCU_APB1_GRP1_TIM2_STOP – LL_DBGMCU_APB1_GRP1_TIM3_STOP (*) – LL_DBGMCU_APB1_GRP1_TIM4_STOP (*) – LL_DBGMCU_APB1_GRP1_TIM5_STOP (*) – LL_DBGMCU_APB1_GRP1_TIM6_STOP – LL_DBGMCU_APB1_GRP1_TIM7_STOP (*) – LL_DBGMCU_APB1_GRP1_RTC_STOP – LL_DBGMCU_APB1_GRP1_WWDG_STOP – LL_DBGMCU_APB1_GRP1_IWDG_STOP – LL_DBGMCU_APB1_GRP1_I2C1_STOP – LL_DBGMCU_APB1_GRP1_I2C2_STOP (*) – LL_DBGMCU_APB1_GRP1_I2C3_STOP – LL_DBGMCU_APB1_GRP1_CAN_STOP – LL_DBGMCU_APB1_GRP1_CAN2_STOP (*) – LL_DBGMCU_APB1_GRP1_LPTIM1_STOP
Return values	• **None:**
Reference Manual to LL API cross	• DBGMCU_APB1FZR1 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph

reference:

LL_DBGMCU_APB1_GRP2_FreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_FreezePeriph (uint32_t Periphs)**
Function description	Freeze APB1 peripherals (group2 peripherals)
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_DBGMCU_APB1_GRP2_I2C4_STOP (*) – LL_DBGMCU_APB1_GRP2_LPTIM2_STOP
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_APB1FZR2 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph

LL_DBGMCU_APB1_GRP1_UnFreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_UnFreezePeriph (uint32_t Periphs)**
Function description	Unfreeze APB1 peripherals (group1 peripherals)
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_DBGMCU_APB1_GRP1_TIM2_STOP – LL_DBGMCU_APB1_GRP1_TIM3_STOP (*) – LL_DBGMCU_APB1_GRP1_TIM4_STOP (*) – LL_DBGMCU_APB1_GRP1_TIM5_STOP (*) – LL_DBGMCU_APB1_GRP1_TIM6_STOP – LL_DBGMCU_APB1_GRP1_TIM7_STOP (*) – LL_DBGMCU_APB1_GRP1_RTC_STOP – LL_DBGMCU_APB1_GRP1_WWDG_STOP – LL_DBGMCU_APB1_GRP1_IWDG_STOP – LL_DBGMCU_APB1_GRP1_I2C1_STOP – LL_DBGMCU_APB1_GRP1_I2C2_STOP (*) – LL_DBGMCU_APB1_GRP1_I2C3_STOP – LL_DBGMCU_APB1_GRP1_CAN_STOP – LL_DBGMCU_APB1_GRP1_CAN2_STOP (*) – LL_DBGMCU_APB1_GRP1_LPTIM1_STOP
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_APB1FZR1 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph

LL_DBGMCU_APB1_GRP2_UnFreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_UnFreezePeriph (uint32_t Periphs)**

Function description	Unfreeze APB1 peripherals (group2 peripherals)
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_DBGMCU_APB1_GRP2_I2C4_STOP (*) – LL_DBGMCU_APB1_GRP2_LPTIM2_STOP
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_APB1FZR2 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph

LL_DBGMCU_APB2_GRP1_FreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_FreezePeriph (uint32_t Periphs)**
Function description	Freeze APB2 peripherals.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_DBGMCU_APB2_GRP1_TIM1_STOP – LL_DBGMCU_APB2_GRP1_TIM8_STOP (*) – LL_DBGMCU_APB2_GRP1_TIM15_STOP – LL_DBGMCU_APB2_GRP1_TIM16_STOP – LL_DBGMCU_APB2_GRP1_TIM17_STOP (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_APB2FZ DBG_TIMx_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph

LL_DBGMCU_APB2_GRP1_UnFreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_UnFreezePeriph (uint32_t Periphs)**
Function description	Unfreeze APB2 peripherals.
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_DBGMCU_APB2_GRP1_TIM1_STOP – LL_DBGMCU_APB2_GRP1_TIM8_STOP (*) – LL_DBGMCU_APB2_GRP1_TIM15_STOP – LL_DBGMCU_APB2_GRP1_TIM16_STOP – LL_DBGMCU_APB2_GRP1_TIM17_STOP (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_APB2FZ DBG_TIMx_STOP LL_DBGMCU_APB2_GRP1_UnFreezePeriph

LL_VREFBUF_Enable

Function name	**__STATIC_INLINE void LL_VREFBUF_Enable (void)**

Function description	Enable Internal voltage reference.
Return values	• **None:**
Reference Manual to LL API cross reference:	• VREFBUF_CSR ENVR LL_VREFBUF_Enable

LL_VREFBUF_Disable

Function name	**__STATIC_INLINE void LL_VREFBUF_Disable (void)**
Function description	Disable Internal voltage reference.
Return values	• **None:**
Reference Manual to LL API cross reference:	• VREFBUF_CSR ENVR LL_VREFBUF_Disable

LL_VREFBUF_EnableHIZ

Function name	**__STATIC_INLINE void LL_VREFBUF_EnableHIZ (void)**
Function description	Enable high impedance (VREF+pin is high impedance)
Return values	• **None:**
Reference Manual to LL API cross reference:	• VREFBUF_CSR HIZ LL_VREFBUF_EnableHIZ

LL_VREFBUF_DisableHIZ

Function name	**__STATIC_INLINE void LL_VREFBUF_DisableHIZ (void)**
Function description	Disable high impedance (VREF+pin is internally connected to the voltage reference buffer output)
Return values	• **None:**
Reference Manual to LL API cross reference:	• VREFBUF_CSR HIZ LL_VREFBUF_DisableHIZ

LL_VREFBUF_SetVoltageScaling

Function name	**__STATIC_INLINE void LL_VREFBUF_SetVoltageScaling (uint32_t Scale)**
Function description	Set the Voltage reference scale.
Parameters	• **Scale:** This parameter can be one of the following values: – LL_VREFBUF_VOLTAGE_SCALE0 – LL_VREFBUF_VOLTAGE_SCALE1
Return values	• **None:**
Reference Manual to LL API cross reference:	• VREFBUF_CSR VRS LL_VREFBUF_SetVoltageScaling

LL_VREFBUF_GetVoltageScaling

Function name	**__STATIC_INLINE uint32_t LL_VREFBUF_GetVoltageScaling (void)**
Function description	Get the Voltage reference scale.
Return values	• **Returned:** value can be one of the following values: – LL_VREFBUF_VOLTAGE_SCALE0 – LL_VREFBUF_VOLTAGE_SCALE1
Reference Manual to LL API cross reference:	• VREFBUF_CSR VRS LL_VREFBUF_GetVoltageScaling

LL_VREFBUF_IsVREFReady

Function name	**__STATIC_INLINE uint32_t LL_VREFBUF_IsVREFReady (void)**
Function description	Check if Voltage reference buffer is ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• VREFBUF_CSR VRR LL_VREFBUF_IsVREFReady

LL_VREFBUF_GetTrimming

Function name	**__STATIC_INLINE uint32_t LL_VREFBUF_GetTrimming (void)**
Function description	Get the trimming code for VREFBUF calibration.
Return values	• **Between:** 0 and 0x3F
Reference Manual to LL API cross reference:	• VREFBUF_CCR TRIM LL_VREFBUF_GetTrimming

LL_VREFBUF_SetTrimming

Function name	**__STATIC_INLINE void LL_VREFBUF_SetTrimming (uint32_t Value)**
Function description	Set the trimming code for VREFBUF calibration (Tune the internal reference buffer voltage)
Parameters	• **Value:** Between 0 and 0x3F
Return values	• **None:**
Reference Manual to LL API cross reference:	• VREFBUF_CCR TRIM LL_VREFBUF_SetTrimming

LL_FLASH_SetLatency

Function name	**__STATIC_INLINE void LL_FLASH_SetLatency (uint32_t Latency)**

Function description	Set FLASH Latency.
Parameters	• **Latency:** This parameter can be one of the following values: (*) value not defined in all devices. – LL_FLASH_LATENCY_0 – LL_FLASH_LATENCY_1 – LL_FLASH_LATENCY_2 – LL_FLASH_LATENCY_3 – LL_FLASH_LATENCY_4 – LL_FLASH_LATENCY_5 (*) – LL_FLASH_LATENCY_6 (*) – LL_FLASH_LATENCY_7 (*) – LL_FLASH_LATENCY_8 (*) – LL_FLASH_LATENCY_9 (*) – LL_FLASH_LATENCY_10 (*) – LL_FLASH_LATENCY_11 (*) – LL_FLASH_LATENCY_12 (*) – LL_FLASH_LATENCY_13 (*) – LL_FLASH_LATENCY_14 (*) – LL_FLASH_LATENCY_15 (*)
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR LATENCY LL_FLASH_SetLatency

LL_FLASH_GetLatency

Function name	**__STATIC_INLINE uint32_t LL_FLASH_GetLatency (void)**
Function description	Get FLASH Latency.
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_FLASH_LATENCY_0 – LL_FLASH_LATENCY_1 – LL_FLASH_LATENCY_2 – LL_FLASH_LATENCY_3 – LL_FLASH_LATENCY_4 – LL_FLASH_LATENCY_5 (*) – LL_FLASH_LATENCY_6 (*) – LL_FLASH_LATENCY_7 (*) – LL_FLASH_LATENCY_8 (*) – LL_FLASH_LATENCY_9 (*) – LL_FLASH_LATENCY_10 (*) – LL_FLASH_LATENCY_11 (*) – LL_FLASH_LATENCY_12 (*) – LL_FLASH_LATENCY_13 (*) – LL_FLASH_LATENCY_14 (*) – LL_FLASH_LATENCY_15 (*)
Reference Manual to LL API cross reference:	• FLASH_ACR LATENCY LL_FLASH_GetLatency

LL_FLASH_EnablePrefetch

Function name	**__STATIC_INLINE void LL_FLASH_EnablePrefetch (void)**
Function description	Enable Prefetch.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR PRFTEN LL_FLASH_EnablePrefetch

LL_FLASH_DisablePrefetch

Function name	**__STATIC_INLINE void LL_FLASH_DisablePrefetch (void)**
Function description	Disable Prefetch.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR PRFTEN LL_FLASH_DisablePrefetch

LL_FLASH_IsPrefetchEnabled

Function name	**__STATIC_INLINE uint32_t LL_FLASH_IsPrefetchEnabled (void)**
Function description	Check if Prefetch buffer is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• FLASH_ACR PRFTEN LL_FLASH_IsPrefetchEnabled

LL_FLASH_EnableInstCache

Function name	**__STATIC_INLINE void LL_FLASH_EnableInstCache (void)**
Function description	Enable Instruction cache.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR ICEN LL_FLASH_EnableInstCache

LL_FLASH_DisableInstCache

Function name	**__STATIC_INLINE void LL_FLASH_DisableInstCache (void)**
Function description	Disable Instruction cache.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR ICEN LL_FLASH_DisableInstCache

LL_FLASH_EnableDataCache

Function name	**__STATIC_INLINE void LL_FLASH_EnableDataCache (void)**
Function description	Enable Data cache.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR DCEN LL_FLASH_EnableDataCache

LL_FLASH_DisableDataCache

Function name	**__STATIC_INLINE void LL_FLASH_DisableDataCache (void)**
Function description	Disable Data cache.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR DCEN LL_FLASH_DisableDataCache

LL_FLASH_EnableInstCacheReset

Function name	**__STATIC_INLINE void LL_FLASH_EnableInstCacheReset (void)**
Function description	Enable Instruction cache reset.
Return values	• **None:**
Notes	• bit can be written only when the instruction cache is disabled
Reference Manual to LL API cross reference:	• FLASH_ACR ICRST LL_FLASH_EnableInstCacheReset

LL_FLASH_DisableInstCacheReset

Function name	**__STATIC_INLINE void LL_FLASH_DisableInstCacheReset (void)**
Function description	Disable Instruction cache reset.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR ICRST LL_FLASH_DisableInstCacheReset

LL_FLASH_EnableDataCacheReset

Function name	**__STATIC_INLINE void LL_FLASH_EnableDataCacheReset (void)**
Function description	Enable Data cache reset.
Return values	• **None:**
Notes	• bit can be written only when the data cache is disabled

Reference Manual to LL API cross reference:	• FLASH_ACR DCRST LL_FLASH_EnableDataCacheReset

LL_FLASH_DisableDataCacheReset

Function name	**__STATIC_INLINE void LL_FLASH_DisableDataCacheReset (void)**
Function description	Disable Data cache reset.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR DCRST LL_FLASH_DisableDataCacheReset

LL_FLASH_EnableRunPowerDown

Function name	**__STATIC_INLINE void LL_FLASH_EnableRunPowerDown (void)**
Function description	Enable Flash Power-down mode during run mode or Low-power run mode.
Return values	• **None:**
Notes	• Flash memory can be put in power-down mode only when the code is executed from RAM • Flash must not be accessed when power down is enabled • Flash must not be put in power-down while a program or an erase operation is on-going
Reference Manual to LL API cross reference:	• FLASH_ACR RUN_PD LL_FLASH_EnableRunPowerDown • FLASH_PDKEYR PDKEY1 LL_FLASH_EnableRunPowerDown • FLASH_PDKEYR PDKEY2 LL_FLASH_EnableRunPowerDown

LL_FLASH_DisableRunPowerDown

Function name	**__STATIC_INLINE void LL_FLASH_DisableRunPowerDown (void)**
Function description	Disable Flash Power-down mode during run mode or Low-power run mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR RUN_PD LL_FLASH_DisableRunPowerDown • FLASH_PDKEYR PDKEY1 LL_FLASH_DisableRunPowerDown • FLASH_PDKEYR PDKEY2 LL_FLASH_DisableRunPowerDown

LL_FLASH_EnableSleepPowerDown

Function name	**__STATIC_INLINE void LL_FLASH_EnableSleepPowerDown**

	(void)
Function description	Enable Flash Power-down mode during Sleep or Low-power sleep mode.
Return values	• **None:**
Notes	• Flash must not be put in power-down while a program or an erase operation is on-going
Reference Manual to LL API cross reference:	• FLASH_ACR SLEEP_PD LL_FLASH_EnableSleepPowerDown

LL_FLASH_DisableSleepPowerDown

Function name	**__STATIC_INLINE void LL_FLASH_DisableSleepPowerDown (void)**
Function description	Disable Flash Power-down mode during Sleep or Low-power sleep mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR SLEEP_PD LL_FLASH_DisableSleepPowerDown

95.2 SYSTEM Firmware driver defines

95.2.1 SYSTEM

DBGMCU APB1 GRP1 STOP IP

LL_DBGMCU_APB1_GRP1_TIM2_STOP	The counter clock of TIM2 is stopped when the core is halted
LL_DBGMCU_APB1_GRP1_TIM3_STOP	The counter clock of TIM3 is stopped when the core is halted
LL_DBGMCU_APB1_GRP1_TIM4_STOP	The counter clock of TIM4 is stopped when the core is halted
LL_DBGMCU_APB1_GRP1_TIM5_STOP	The counter clock of TIM5 is stopped when the core is halted
LL_DBGMCU_APB1_GRP1_TIM6_STOP	The counter clock of TIM6 is stopped when the core is halted
LL_DBGMCU_APB1_GRP1_TIM7_STOP	The counter clock of TIM7 is stopped when the core is halted
LL_DBGMCU_APB1_GRP1_RTC_STOP	The clock of the RTC counter is stopped when the core is halted
LL_DBGMCU_APB1_GRP1_WWDG_STOP	The window watchdog counter clock is stopped when the core is halted
LL_DBGMCU_APB1_GRP1_IWDG_STOP	The independent watchdog counter clock is stopped when the core is halted
LL_DBGMCU_APB1_GRP1_I2C1_STOP	The I2C1 SMBus timeout is frozen

LL_DBGMCU_APB1_GRP1_I2C2_STOP	The I2C2 SMBus timeout is frozen
LL_DBGMCU_APB1_GRP1_I2C3_STOP	The I2C3 SMBus timeout is frozen
LL_DBGMCU_APB1_GRP1_CAN_STOP	The bxCAN receive registers are frozen
LL_DBGMCU_APB1_GRP1_LPTIM1_STOP	The counter clock of LPTIM1 is stopped when the core is halted

DBGMCU APB1 GRP2 STOP IP

LL_DBGMCU_APB1_GRP2_I2C4_STOP	The I2C4 SMBus timeout is frozen
LL_DBGMCU_APB1_GRP2_LPTIM2_STOP	The counter clock of LPTIM2 is stopped when the core is halted

DBGMCU APB2 GRP1 STOP IP

LL_DBGMCU_APB2_GRP1_TIM1_STOP	The counter clock of TIM1 is stopped when the core is halted
LL_DBGMCU_APB2_GRP1_TIM8_STOP	The counter clock of TIM8 is stopped when the core is halted
LL_DBGMCU_APB2_GRP1_TIM15_STOP	The counter clock of TIM15 is stopped when the core is halted
LL_DBGMCU_APB2_GRP1_TIM16_STOP	The counter clock of TIM16 is stopped when the core is halted
LL_DBGMCU_APB2_GRP1_TIM17_STOP	The counter clock of TIM17 is stopped when the core is halted

SYSCFG BANK MODE

LL_SYSCFG_BANKMODE_BANK1	Flash Bank1 mapped at 0x08000000 (and aliased @0x00000000) and Flash Bank2 mapped at 0x08080000 (and aliased at 0x00080000)
LL_SYSCFG_BANKMODE_BANK2	Flash Bank2 mapped at 0x08000000 (and aliased @0x00000000) and Flash Bank1 mapped at 0x08080000 (and aliased at 0x00080000)

SYSCFG EXTI LINE

LL_SYSCFG_EXTI_LINE0

LL_SYSCFG_EXTI_LINE1

LL_SYSCFG_EXTI_LINE2

LL_SYSCFG_EXTI_LINE3

LL_SYSCFG_EXTI_LINE4

LL_SYSCFG_EXTI_LINE5

LL_SYSCFG_EXTI_LINE6

LL_SYSCFG_EXTI_LINE7

LL_SYSCFG_EXTI_LINE8

LL_SYSCFG_EXTI_LINE9

LL_SYSCFG_EXTI_LINE10

LL_SYSCFG_EXTI_LINE11

LL_SYSCFG_EXTI_LINE12

LL_SYSCFG_EXTI_LINE13

LL_SYSCFG_EXTI_LINE14

LL_SYSCFG_EXTI_LINE15

SYSCFG EXTI PORT

LL_SYSCFG_EXTI_PORTA	EXTI PORT A
LL_SYSCFG_EXTI_PORTB	EXTI PORT B
LL_SYSCFG_EXTI_PORTC	EXTI PORT C
LL_SYSCFG_EXTI_PORTD	EXTI PORT D
LL_SYSCFG_EXTI_PORTE	EXTI PORT E
LL_SYSCFG_EXTI_PORTF	EXTI PORT F
LL_SYSCFG_EXTI_PORTG	EXTI PORT G
LL_SYSCFG_EXTI_PORTH	EXTI PORT H
LL_SYSCFG_EXTI_PORTI	EXTI PORT I

SYSCFG I2C FASTMODEPLUS

LL_SYSCFG_I2C_FASTMODEPLUS_PB6	Enable Fast Mode Plus on PB6
LL_SYSCFG_I2C_FASTMODEPLUS_PB7	Enable Fast Mode Plus on PB7
LL_SYSCFG_I2C_FASTMODEPLUS_PB8	Enable Fast Mode Plus on PB8
LL_SYSCFG_I2C_FASTMODEPLUS_PB9	Enable Fast Mode Plus on PB9
LL_SYSCFG_I2C_FASTMODEPLUS_I2C1	Enable Fast Mode Plus on I2C1 pins
LL_SYSCFG_I2C_FASTMODEPLUS_I2C2	Enable Fast Mode Plus on I2C2 pins
LL_SYSCFG_I2C_FASTMODEPLUS_I2C3	Enable Fast Mode Plus on I2C3 pins
LL_SYSCFG_I2C_FASTMODEPLUS_I2C4	Enable Fast Mode Plus on I2C4 pins

FLASH LATENCY

LL_FLASH_LATENCY_0	FLASH Zero wait state
LL_FLASH_LATENCY_1	FLASH One wait state
LL_FLASH_LATENCY_2	FLASH Two wait states
LL_FLASH_LATENCY_3	FLASH Three wait states
LL_FLASH_LATENCY_4	FLASH Four wait states
LL_FLASH_LATENCY_5	FLASH five wait state
LL_FLASH_LATENCY_6	FLASH six wait state
LL_FLASH_LATENCY_7	FLASH seven wait states
LL_FLASH_LATENCY_8	FLASH eight wait states
LL_FLASH_LATENCY_9	FLASH nine wait states
LL_FLASH_LATENCY_10	FLASH ten wait states
LL_FLASH_LATENCY_11	FLASH eleven wait states

LL_FLASH_LATENCY_12	FLASH twelve wait states
LL_FLASH_LATENCY_13	FLASH thirteen wait states
LL_FLASH_LATENCY_14	FLASH fourteen wait states
LL_FLASH_LATENCY_15	FLASH fifteen wait states

SYSCFG REMAP

LL_SYSCFG_REMAP_FLASH	Main Flash memory mapped at 0x00000000
LL_SYSCFG_REMAP_SYSTEMFLASH	System Flash memory mapped at 0x00000000
LL_SYSCFG_REMAP_SRAM	SRAM1 mapped at 0x00000000
LL_SYSCFG_REMAP_FMC	FMC bank 1 (NOR/PSRAM 1 and 2) mapped at 0x00000000
LL_SYSCFG_REMAP_QUADSPI	QUADSPI memory mapped at 0x00000000

SYSCFG SRAM2 WRP

LL_SYSCFG_SRAM2WRP_PAGE0	SRAM2 Write protection page 0
LL_SYSCFG_SRAM2WRP_PAGE1	SRAM2 Write protection page 1
LL_SYSCFG_SRAM2WRP_PAGE2	SRAM2 Write protection page 2
LL_SYSCFG_SRAM2WRP_PAGE3	SRAM2 Write protection page 3
LL_SYSCFG_SRAM2WRP_PAGE4	SRAM2 Write protection page 4
LL_SYSCFG_SRAM2WRP_PAGE5	SRAM2 Write protection page 5
LL_SYSCFG_SRAM2WRP_PAGE6	SRAM2 Write protection page 6
LL_SYSCFG_SRAM2WRP_PAGE7	SRAM2 Write protection page 7
LL_SYSCFG_SRAM2WRP_PAGE8	SRAM2 Write protection page 8
LL_SYSCFG_SRAM2WRP_PAGE9	SRAM2 Write protection page 9
LL_SYSCFG_SRAM2WRP_PAGE10	SRAM2 Write protection page 10
LL_SYSCFG_SRAM2WRP_PAGE11	SRAM2 Write protection page 11
LL_SYSCFG_SRAM2WRP_PAGE12	SRAM2 Write protection page 12
LL_SYSCFG_SRAM2WRP_PAGE13	SRAM2 Write protection page 13
LL_SYSCFG_SRAM2WRP_PAGE14	SRAM2 Write protection page 14
LL_SYSCFG_SRAM2WRP_PAGE15	SRAM2 Write protection page 15
LL_SYSCFG_SRAM2WRP_PAGE16	SRAM2 Write protection page 16
LL_SYSCFG_SRAM2WRP_PAGE17	SRAM2 Write protection page 17
LL_SYSCFG_SRAM2WRP_PAGE18	SRAM2 Write protection page 18
LL_SYSCFG_SRAM2WRP_PAGE19	SRAM2 Write protection page 19
LL_SYSCFG_SRAM2WRP_PAGE20	SRAM2 Write protection page 20
LL_SYSCFG_SRAM2WRP_PAGE21	SRAM2 Write protection page 21
LL_SYSCFG_SRAM2WRP_PAGE22	SRAM2 Write protection page 22
LL_SYSCFG_SRAM2WRP_PAGE23	SRAM2 Write protection page 23
LL_SYSCFG_SRAM2WRP_PAGE24	SRAM2 Write protection page 24

LL_SYSCFG_SRAM2WRP_PAGE25	SRAM2 Write protection page 25
LL_SYSCFG_SRAM2WRP_PAGE26	SRAM2 Write protection page 26
LL_SYSCFG_SRAM2WRP_PAGE27	SRAM2 Write protection page 27
LL_SYSCFG_SRAM2WRP_PAGE28	SRAM2 Write protection page 28
LL_SYSCFG_SRAM2WRP_PAGE29	SRAM2 Write protection page 29
LL_SYSCFG_SRAM2WRP_PAGE30	SRAM2 Write protection page 30
LL_SYSCFG_SRAM2WRP_PAGE31	SRAM2 Write protection page 31
LL_SYSCFG_SRAM2WRP_PAGE32	SRAM2 Write protection page 32
LL_SYSCFG_SRAM2WRP_PAGE33	SRAM2 Write protection page 33
LL_SYSCFG_SRAM2WRP_PAGE34	SRAM2 Write protection page 34
LL_SYSCFG_SRAM2WRP_PAGE35	SRAM2 Write protection page 35
LL_SYSCFG_SRAM2WRP_PAGE36	SRAM2 Write protection page 36
LL_SYSCFG_SRAM2WRP_PAGE37	SRAM2 Write protection page 37
LL_SYSCFG_SRAM2WRP_PAGE38	SRAM2 Write protection page 38
LL_SYSCFG_SRAM2WRP_PAGE39	SRAM2 Write protection page 39
LL_SYSCFG_SRAM2WRP_PAGE40	SRAM2 Write protection page 40
LL_SYSCFG_SRAM2WRP_PAGE41	SRAM2 Write protection page 41
LL_SYSCFG_SRAM2WRP_PAGE42	SRAM2 Write protection page 42
LL_SYSCFG_SRAM2WRP_PAGE43	SRAM2 Write protection page 43
LL_SYSCFG_SRAM2WRP_PAGE44	SRAM2 Write protection page 44
LL_SYSCFG_SRAM2WRP_PAGE45	SRAM2 Write protection page 45
LL_SYSCFG_SRAM2WRP_PAGE46	SRAM2 Write protection page 46
LL_SYSCFG_SRAM2WRP_PAGE47	SRAM2 Write protection page 47
LL_SYSCFG_SRAM2WRP_PAGE48	SRAM2 Write protection page 48
LL_SYSCFG_SRAM2WRP_PAGE49	SRAM2 Write protection page 49
LL_SYSCFG_SRAM2WRP_PAGE50	SRAM2 Write protection page 50
LL_SYSCFG_SRAM2WRP_PAGE51	SRAM2 Write protection page 51
LL_SYSCFG_SRAM2WRP_PAGE52	SRAM2 Write protection page 52
LL_SYSCFG_SRAM2WRP_PAGE53	SRAM2 Write protection page 53
LL_SYSCFG_SRAM2WRP_PAGE54	SRAM2 Write protection page 54
LL_SYSCFG_SRAM2WRP_PAGE55	SRAM2 Write protection page 55
LL_SYSCFG_SRAM2WRP_PAGE56	SRAM2 Write protection page 56
LL_SYSCFG_SRAM2WRP_PAGE57	SRAM2 Write protection page 57
LL_SYSCFG_SRAM2WRP_PAGE58	SRAM2 Write protection page 58
LL_SYSCFG_SRAM2WRP_PAGE59	SRAM2 Write protection page 59
LL_SYSCFG_SRAM2WRP_PAGE60	SRAM2 Write protection page 60

LL_SYSCFG_SRAM2WRP_PAGE61	SRAM2 Write protection page 61
LL_SYSCFG_SRAM2WRP_PAGE62	SRAM2 Write protection page 62
LL_SYSCFG_SRAM2WRP_PAGE63	SRAM2 Write protection page 63

SYSCFG TIMER BREAK

LL_SYSCFG_TIMBREAK_ECC	Enables and locks the ECC error signal with Break Input of TIM1/8/15/16/17
LL_SYSCFG_TIMBREAK_PVD	Enables and locks the PVD connection with TIM1/8/15/16/17 Break Input and also the PVDE and PLS bits of the Power Control Interface
LL_SYSCFG_TIMBREAK_SRAM2_PARITY	Enables and locks the SRAM2_PARITY error signal with Break Input of TIM1/8/15/16/17
LL_SYSCFG_TIMBREAK_LOCKUP	Enables and locks the LOCKUP output of CortexM4 with Break Input of TIM1/15/16/17

DBGMCU TRACE Pin Assignment

LL_DBGMCU_TRACE_NONE	TRACE pins not assigned (default state)
LL_DBGMCU_TRACE_ASYNCH	TRACE pin assignment for Asynchronous Mode
LL_DBGMCU_TRACE_SYNCH_SIZE1	TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 1
LL_DBGMCU_TRACE_SYNCH_SIZE2	TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 2
LL_DBGMCU_TRACE_SYNCH_SIZE4	TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 4

VREFBUF VOLTAGE

LL_VREFBUF_VOLTAGE_SCALE0	Voltage reference scale 0 (VREF_OUT1)
LL_VREFBUF_VOLTAGE_SCALE1	Voltage reference scale 1 (VREF_OUT2)

SYSCFG

LL_SYSCFG_EnableSRAM2PageWRP

Description:

- Enable SRAM2 page write protection for Pages in range 0 to 31.

Parameters:

- SRAM2WRP: This parameter can be a combination of the following values:
  - LL_SYSCFG_SRAM2WRP_PAGE0
  - LL_SYSCFG_SRAM2WRP_PAGE1
  - LL_SYSCFG_SRAM2WRP_PAGE2
  - LL_SYSCFG_SRAM2WRP_PAGE3
  - LL_SYSCFG_SRAM2WRP_PAGE4
  - LL_SYSCFG_SRAM2WRP_PAGE5
  - LL_SYSCFG_SRAM2WRP_PAGE6
  - LL_SYSCFG_SRAM2WRP_PAGE7
  - LL_SYSCFG_SRAM2WRP_PAGE8
  - LL_SYSCFG_SRAM2WRP_PAGE9

- LL_SYSCFG_SRAM2WRP_PAGE10
- LL_SYSCFG_SRAM2WRP_PAGE11
- LL_SYSCFG_SRAM2WRP_PAGE12
- LL_SYSCFG_SRAM2WRP_PAGE13
- LL_SYSCFG_SRAM2WRP_PAGE14
- LL_SYSCFG_SRAM2WRP_PAGE15
- LL_SYSCFG_SRAM2WRP_PAGE16 (*)
- LL_SYSCFG_SRAM2WRP_PAGE17 (*)
- LL_SYSCFG_SRAM2WRP_PAGE18 (*)
- LL_SYSCFG_SRAM2WRP_PAGE19 (*)
- LL_SYSCFG_SRAM2WRP_PAGE20 (*)
- LL_SYSCFG_SRAM2WRP_PAGE21 (*)
- LL_SYSCFG_SRAM2WRP_PAGE22 (*)
- LL_SYSCFG_SRAM2WRP_PAGE23 (*)
- LL_SYSCFG_SRAM2WRP_PAGE24 (*)
- LL_SYSCFG_SRAM2WRP_PAGE25 (*)
- LL_SYSCFG_SRAM2WRP_PAGE26 (*)
- LL_SYSCFG_SRAM2WRP_PAGE27 (*)
- LL_SYSCFG_SRAM2WRP_PAGE28 (*)
- LL_SYSCFG_SRAM2WRP_PAGE29 (*)
- LL_SYSCFG_SRAM2WRP_PAGE30 (*)
- LL_SYSCFG_SRAM2WRP_PAGE31 (*)

Return value:

- None

Notes:

- Write protection is cleared only by a system reset

96 LL TIM Generic Driver

96.1 TIM Firmware driver registers structures

96.1.1 LL_TIM_InitTypeDef

Data Fields

- ***uint16_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Autoreload***
- ***uint32_t ClockDivision***
- ***uint8_t RepetitionCounter***

Field Documentation

- ***uint16_t LL_TIM_InitTypeDef::Prescaler***
  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetPrescaler()**.
- ***uint32_t LL_TIM_InitTypeDef::CounterMode***
  Specifies the counter mode. This parameter can be a value of ***TIM_LL_EC_COUNTERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetCounterMode()**.
- ***uint32_t LL_TIM_InitTypeDef::Autoreload***
  Specifies the auto reload value to be loaded into the active Auto-Reload Register at the next update event. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF. Some timer instances may support 32 bits counters. In that case this parameter must be a number between 0x0000 and 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetAutoReload()**.
- ***uint32_t LL_TIM_InitTypeDef::ClockDivision***
  Specifies the clock division. This parameter can be a value of ***TIM_LL_EC_CLOCKDIVISION***.This feature can be modified afterwards using unitary function **LL_TIM_SetClockDivision()**.
- ***uint8_t LL_TIM_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:the number of PWM periods in edge-aligned modethe number of half PWM period in center-aligned mode This parameter must be a number between 0x00 and 0xFF. This feature can be modified afterwards using unitary function **LL_TIM_SetRepetitionCounter()**.

96.1.2 LL_TIM_OC_InitTypeDef

Data Fields

- ***uint32_t OCMode***
- ***uint32_t OCState***
- ***uint32_t OCNState***
- ***uint32_t CompareValue***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t LL_TIM_OC_InitTypeDef::OCMode***
  Specifies the output mode. This parameter can be a value of ***TIM_LL_EC_OCMODE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetMode()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCState***
  Specifies the TIM Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNState***
  Specifies the TIM complementary Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::CompareValue***
  Specifies the Compare value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function LL_TIM_OC_SetCompareCHx (x=1..6).
- ***uint32_t LL_TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.

96.1.3 LL_TIM_IC_InitTypeDef

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICActiveInput***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t LL_TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICActiveInput***
  Specifies the input. This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of

TIM_LL_EC_ICPSC.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICFilter***
Specifies the input capture filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

96.1.4 LL_TIM_ENCODER_InitTypeDef

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1ActiveInput***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2ActiveInput***
- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::EncoderMode***
Specifies the encoder resolution (x2 or x4). This parameter can be a value of ***TIM_LL_EC_ENCODERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetEncoderMode()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Polarity***
Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1ActiveInput***
Specifies the TI1 input source This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Prescaler***
Specifies the TI1 input prescaler value. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Filter***
Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Polarity***
Specifies the active edge of TI2 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2ActiveInput***
Specifies the TI2 input source This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Prescaler***
Specifies the TI2 input prescaler value. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Filter***
  Specifies the TI2 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

96.1.5 LL_TIM_HALLSENSOR_InitTypeDef

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t CommutationDelay***

Field Documentation

- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Polarity***
  Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Prescaler***
  Specifies the TI1 input prescaler value. Prescaler must be set to get a maximum counter period longer than the time interval between 2 consecutive changes on the Hall inputs. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Filter***
  Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::CommutationDelay***
  Specifies the compare value to be loaded into the Capture Compare Register. A positive pulse (TRGO event) is generated with a programmable delay every time a change occurs on the Hall inputs. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetCompareCH2()**.

96.1.6 LL_TIM_BDTR_InitTypeDef

Data Fields

- ***uint32_t OSSRState***
- ***uint32_t OSSIState***
- ***uint32_t LockLevel***
- ***uint8_t DeadTime***
- ***uint16_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSRState***
  Specifies the Off-State selection used in Run mode. This parameter can be a value of ***TIM_LL_EC_OSSR***This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**

Note: This bit-field cannot be modified as long as LOCK level 2 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSIState***
  Specifies the Off-State used in Idle state. This parameter can be a value of *TIM_LL_EC_OSSI*This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**
  Note: This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::LockLevel***
  Specifies the LOCK level parameters. This parameter can be a value of *TIM_LL_EC_LOCKLEVEL*
  Note: The LOCK bits can be written only once after the reset. Once the TIMx_BDTR register has been written, their content is frozen until the next reset.
- ***uint8_t LL_TIM_BDTR_InitTypeDef::DeadTime***
  Specifies the delay time between the switching-off and the switching-on of the outputs. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetDeadTime()**
  Note: This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed.
- ***uint16_t LL_TIM_BDTR_InitTypeDef::BreakState***
  Specifies whether the TIM Break input is enabled or not. This parameter can be a value of *TIM_LL_EC_BREAK_ENABLE*This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK()** or **LL_TIM_DisableBRK()**
  Note: This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakPolarity***
  Specifies the TIM Break Input pin polarity. This parameter can be a value of *TIM_LL_EC_BREAK_POLARITY*This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**
  Note: This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakFilter***
  Specifies the TIM Break Filter. This parameter can be a value of *TIM_LL_EC_BREAK_FILTER*This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**
  Note: This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2State***
  Specifies whether the TIM Break2 input is enabled or not. This parameter can be a value of *TIM_LL_EC_BREAK2_ENABLE*This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK2()** or **LL_TIM_DisableBRK2()**
  Note: This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Polarity***
  Specifies the TIM Break2 Input pin polarity. This parameter can be a value of *TIM_LL_EC_BREAK2_POLARITY*This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**
  Note: This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Filter***
  Specifies the TIM Break2 Filter. This parameter can be a value of *TIM_LL_EC_BREAK2_FILTER*This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**

Note: This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::AutomaticOutput***
  Specifies whether the TIM Automatic Output feature is enabled or not. This parameter can be a value of ***TIM_LL_EC_AUTOMATICOUTPUT_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableAutomaticOutput()** or **LL_TIM_DisableAutomaticOutput()**
  Note: This bit-field can not be modified as long as LOCK level 1 has been programmed.

96.2 TIM Firmware driver API description

96.2.1 Detailed description of functions

LL_TIM_EnableCounter

Function name	**__STATIC_INLINE void LL_TIM_EnableCounter (TIM_TypeDef * TIMx)**
Function description	Enable timer counter.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 CEN LL_TIM_EnableCounter

LL_TIM_DisableCounter

Function name	**__STATIC_INLINE void LL_TIM_DisableCounter (TIM_TypeDef * TIMx)**
Function description	Disable timer counter.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 CEN LL_TIM_DisableCounter

LL_TIM_IsEnabledCounter

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledCounter (TIM_TypeDef * TIMx)**
Function description	Indicates whether the timer counter is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 CEN LL_TIM_IsEnabledCounter

LL_TIM_EnableUpdateEvent

Function name	**__STATIC_INLINE void LL_TIM_EnableUpdateEvent (TIM_TypeDef * TIMx)**
Function description	Enable update event generation.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 UDIS LL_TIM_EnableUpdateEvent

LL_TIM_DisableUpdateEvent

Function name	**__STATIC_INLINE void LL_TIM_DisableUpdateEvent (TIM_TypeDef * TIMx)**
Function description	Disable update event generation.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 UDIS LL_TIM_DisableUpdateEvent

LL_TIM_IsEnabledUpdateEvent

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledUpdateEvent (TIM_TypeDef * TIMx)**
Function description	Indicates whether update event generation is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **Inverted:** state of bit (0 or 1).
Reference Manual to LL API cross reference:	• CR1 UDIS LL_TIM_IsEnabledUpdateEvent

LL_TIM_SetUpdateSource

Function name	**__STATIC_INLINE void LL_TIM_SetUpdateSource (TIM_TypeDef * TIMx, uint32_t UpdateSource)**
Function description	Set update event source.
Parameters	• **TIMx:** Timer instance • **UpdateSource:** This parameter can be one of the following values: – LL_TIM_UPDATESOURCE_REGULAR – LL_TIM_UPDATESOURCE_COUNTER
Return values	• **None:**
Notes	• Update event source set to LL_TIM_UPDATESOURCE_REGULAR: any of the following

events generate an update interrupt or DMA request if enabled: Counter overflow/underflowSetting the UG bitUpdate generation through the slave mode controller
- Update event source set to LL_TIM_UPDATESOURCE_COUNTER: only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Reference Manual to LL API cross reference:
- CR1 URS LL_TIM_SetUpdateSource

LL_TIM_GetUpdateSource

Function name: **__STATIC_INLINE uint32_t LL_TIM_GetUpdateSource (TIM_TypeDef * TIMx)**

Function description: Get actual event update source.

Parameters:
- **TIMx:** Timer instance

Return values:
- **Returned:** value can be one of the following values:
  - LL_TIM_UPDATESOURCE_REGULAR
  - LL_TIM_UPDATESOURCE_COUNTER

Reference Manual to LL API cross reference:
- CR1 URS LL_TIM_GetUpdateSource

LL_TIM_SetOnePulseMode

Function name: **__STATIC_INLINE void LL_TIM_SetOnePulseMode (TIM_TypeDef * TIMx, uint32_t OnePulseMode)**

Function description: Set one pulse mode (one shot v.s.

Parameters:
- **TIMx:** Timer instance
- **OnePulseMode:** This parameter can be one of the following values:
  - LL_TIM_ONEPULSEMODE_SINGLE
  - LL_TIM_ONEPULSEMODE_REPETITIVE

Return values:
- **None:**

Reference Manual to LL API cross reference:
- CR1 OPM LL_TIM_SetOnePulseMode

LL_TIM_GetOnePulseMode

Function name: **__STATIC_INLINE uint32_t LL_TIM_GetOnePulseMode (TIM_TypeDef * TIMx)**

Function description: Get actual one pulse mode.

Parameters:
- **TIMx:** Timer instance

Return values:
- **Returned:** value can be one of the following values:
  - LL_TIM_ONEPULSEMODE_SINGLE

– LL_TIM_ONEPULSEMODE_REPETITIVE

Reference Manual to LL API cross reference:
- CR1 OPM LL_TIM_GetOnePulseMode

LL_TIM_SetCounterMode

Function name	**__STATIC_INLINE void LL_TIM_SetCounterMode (TIM_TypeDef * TIMx, uint32_t CounterMode)**
Function description	Set the timer counter counting mode.
Parameters	• **TIMx:** Timer instance • **CounterMode:** This parameter can be one of the following values: – LL_TIM_COUNTERMODE_UP – LL_TIM_COUNTERMODE_DOWN – LL_TIM_COUNTERMODE_CENTER_UP – LL_TIM_COUNTERMODE_CENTER_DOWN – LL_TIM_COUNTERMODE_CENTER_UP_DOWN
Return values	• **None:**
Notes	• Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.
Reference Manual to LL API cross reference:	• CR1 DIR LL_TIM_SetCounterMode • CR1 CMS LL_TIM_SetCounterMode

LL_TIM_GetCounterMode

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetCounterMode (TIM_TypeDef * TIMx)**
Function description	Get actual counter mode.
Parameters	• **TIMx:** Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_TIM_COUNTERMODE_UP – LL_TIM_COUNTERMODE_DOWN – LL_TIM_COUNTERMODE_CENTER_UP – LL_TIM_COUNTERMODE_CENTER_DOWN – LL_TIM_COUNTERMODE_CENTER_UP_DOWN
Notes	• Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.
Reference Manual to LL API cross reference:	• CR1 DIR LL_TIM_GetCounterMode • CR1 CMS LL_TIM_GetCounterMode

LL_TIM_EnableARRPreload

Function name	**__STATIC_INLINE void LL_TIM_EnableARRPreload (TIM_TypeDef * TIMx)**
Function description	Enable auto-reload (ARR) preload.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 ARPE LL_TIM_EnableARRPreload

LL_TIM_DisableARRPreload

Function name	**__STATIC_INLINE void LL_TIM_DisableARRPreload (TIM_TypeDef * TIMx)**
Function description	Disable auto-reload (ARR) preload.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 ARPE LL_TIM_DisableARRPreload

LL_TIM_IsEnabledARRPreload

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledARRPreload (TIM_TypeDef * TIMx)**
Function description	Indicates whether auto-reload (ARR) preload is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 ARPE LL_TIM_IsEnabledARRPreload

LL_TIM_SetClockDivision

Function name	**__STATIC_INLINE void LL_TIM_SetClockDivision (TIM_TypeDef * TIMx, uint32_t ClockDivision)**
Function description	Set the division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.
Parameters	• **TIMx:** Timer instance • **ClockDivision:** This parameter can be one of the following values: – LL_TIM_CLOCKDIVISION_DIV1 – LL_TIM_CLOCKDIVISION_DIV2 – LL_TIM_CLOCKDIVISION_DIV4

Return values	• **None:**
Notes	• Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.
Reference Manual to LL API cross reference:	• CR1 CKD LL_TIM_SetClockDivision

LL_TIM_GetClockDivision

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetClockDivision (TIM_TypeDef * TIMx)**
Function description	Get the actual division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.
Parameters	• **TIMx:** Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_TIM_CLOCKDIVISION_DIV1 – LL_TIM_CLOCKDIVISION_DIV2 – LL_TIM_CLOCKDIVISION_DIV4
Notes	• Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.
Reference Manual to LL API cross reference:	• CR1 CKD LL_TIM_GetClockDivision

LL_TIM_SetCounter

Function name	**__STATIC_INLINE void LL_TIM_SetCounter (TIM_TypeDef * TIMx, uint32_t Counter)**
Function description	Set the counter value.
Parameters	• **TIMx:** Timer instance • **Counter:** Counter value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)
Return values	• **None:**
Notes	• Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
Reference Manual to LL API cross reference:	• CNT CNT LL_TIM_SetCounter

LL_TIM_GetCounter

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetCounter (TIM_TypeDef * TIMx)**

Function description	Get the counter value.
Parameters	• **TIMx:** Timer instance
Return values	• **Counter:** value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)
Notes	• Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
Reference Manual to LL API cross reference:	• CNT CNT LL_TIM_GetCounter

LL_TIM_GetDirection

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetDirection (TIM_TypeDef * TIMx)**
Function description	Get the current direction of the counter.
Parameters	• **TIMx:** Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_TIM_COUNTERDIRECTION_UP – LL_TIM_COUNTERDIRECTION_DOWN
Reference Manual to LL API cross reference:	• CR1 DIR LL_TIM_GetDirection

LL_TIM_SetPrescaler

Function name	**__STATIC_INLINE void LL_TIM_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Prescaler)**
Function description	Set the prescaler value.
Parameters	• **TIMx:** Timer instance • **Prescaler:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**
Notes	• The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1). • The prescaler can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event. • Helper macro __LL_TIM_CALC_PSC can be used to calculate the Prescaler parameter
Reference Manual to LL API cross reference:	• PSC PSC LL_TIM_SetPrescaler

LL_TIM_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetPrescaler (TIM_TypeDef * TIMx)**

Function description	Get the prescaler value.
Parameters	• **TIMx:** Timer instance
Return values	• **Prescaler:** value between Min_Data=0 and Max_Data=65535
Reference Manual to LL API cross reference:	• PSC PSC LL_TIM_GetPrescaler

LL_TIM_SetAutoReload

Function name	**__STATIC_INLINE void LL_TIM_SetAutoReload (TIM_TypeDef * TIMx, uint32_t AutoReload)**
Function description	Set the auto-reload value.
Parameters	• **TIMx:** Timer instance • **AutoReload:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**
Notes	• The counter is blocked while the auto-reload value is null. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Helper macro __LL_TIM_CALC_ARR can be used to calculate the AutoReload parameter
Reference Manual to LL API cross reference:	• ARR ARR LL_TIM_SetAutoReload

LL_TIM_GetAutoReload

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetAutoReload (TIM_TypeDef * TIMx)**
Function description	Get the auto-reload value.
Parameters	• **TIMx:** Timer instance
Return values	• **Auto-reload:** value
Notes	• Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
Reference Manual to LL API cross reference:	• ARR ARR LL_TIM_GetAutoReload

LL_TIM_SetRepetitionCounter

Function name	**__STATIC_INLINE void LL_TIM_SetRepetitionCounter (TIM_TypeDef * TIMx, uint32_t RepetitionCounter)**
Function description	Set the repetition counter value.
Parameters	• **TIMx:** Timer instance

	• **RepetitionCounter:** between Min_Data=0 and Max_Data=255
Return values	• **None:**
Notes	• For advanced timer instances RepetitionCounter can be up to 65535. • Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.
Reference Manual to LL API cross reference:	• RCR REP LL_TIM_SetRepetitionCounter

LL_TIM_GetRepetitionCounter

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetRepetitionCounter (TIM_TypeDef * TIMx)**
Function description	Get the repetition counter value.
Parameters	• **TIMx:** Timer instance
Return values	• **Repetition:** counter value
Notes	• Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.
Reference Manual to LL API cross reference:	• RCR REP LL_TIM_GetRepetitionCounter

LL_TIM_EnableUIFRemap

Function name	**__STATIC_INLINE void LL_TIM_EnableUIFRemap (TIM_TypeDef * TIMx)**
Function description	Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.
Reference Manual to LL API cross reference:	• CR1 UIFREMAP LL_TIM_EnableUIFRemap

LL_TIM_DisableUIFRemap

Function name	**__STATIC_INLINE void LL_TIM_DisableUIFRemap (TIM_TypeDef * TIMx)**
Function description	Disable update interrupt flag (UIF) remapping.

Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 UIFREMAP LL_TIM_DisableUIFRemap

LL_TIM_CC_EnablePreload

Function name	**__STATIC_INLINE void LL_TIM_CC_EnablePreload (TIM_TypeDef * TIMx)**
Function description	Enable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a commutation event (COM) occurs. • Only on channels that have a complementary output. • Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.
Reference Manual to LL API cross reference:	• CR2 CCPC LL_TIM_CC_EnablePreload

LL_TIM_CC_DisablePreload

Function name	**__STATIC_INLINE void LL_TIM_CC_DisablePreload (TIM_TypeDef * TIMx)**
Function description	Disable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.
Reference Manual to LL API cross reference:	• CR2 CCPC LL_TIM_CC_DisablePreload

LL_TIM_CC_SetUpdate

Function name	**__STATIC_INLINE void LL_TIM_CC_SetUpdate (TIM_TypeDef * TIMx, uint32_t CCUpdateSource)**
Function description	Set the updated source of the capture/compare control bits (CCxE, CCxNE and OCxM).
Parameters	• **TIMx:** Timer instance

	• **CCUpdateSource:** This parameter can be one of the following values: – LL_TIM_CCUPDATESOURCE_COMG_ONLY – LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI
Return values	• **None:**
Notes	• Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.
Reference Manual to LL API cross reference:	• CR2 CCUS LL_TIM_CC_SetUpdate

LL_TIM_CC_SetDMAReqTrigger

Function name	**__STATIC_INLINE void LL_TIM_CC_SetDMAReqTrigger (TIM_TypeDef * TIMx, uint32_t DMAReqTrigger)**
Function description	Set the trigger of the capture/compare DMA request.
Parameters	• **TIMx:** Timer instance • **DMAReqTrigger:** This parameter can be one of the following values: – LL_TIM_CCDMAREQUEST_CC – LL_TIM_CCDMAREQUEST_UPDATE
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 CCDS LL_TIM_CC_SetDMAReqTrigger

LL_TIM_CC_GetDMAReqTrigger

Function name	**__STATIC_INLINE uint32_t LL_TIM_CC_GetDMAReqTrigger (TIM_TypeDef * TIMx)**
Function description	Get actual trigger of the capture/compare DMA request.
Parameters	• **TIMx:** Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_TIM_CCDMAREQUEST_CC – LL_TIM_CCDMAREQUEST_UPDATE
Reference Manual to LL API cross reference:	• CR2 CCDS LL_TIM_CC_GetDMAReqTrigger

LL_TIM_CC_SetLockLevel

Function name	**__STATIC_INLINE void LL_TIM_CC_SetLockLevel (TIM_TypeDef * TIMx, uint32_t LockLevel)**
Function description	Set the lock level to freeze the configuration of several capture/compare parameters.
Parameters	• **TIMx:** Timer instance

	• **LockLevel:** This parameter can be one of the following values: – LL_TIM_LOCKLEVEL_OFF – LL_TIM_LOCKLEVEL_1 – LL_TIM_LOCKLEVEL_2 – LL_TIM_LOCKLEVEL_3
Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not the lock mechanism is supported by a timer instance.
Reference Manual to LL API cross reference:	• BDTR LOCK LL_TIM_CC_SetLockLevel

LL_TIM_CC_EnableChannel

Function name	**__STATIC_INLINE void LL_TIM_CC_EnableChannel (TIM_TypeDef * TIMx, uint32_t Channels)**
Function description	Enable capture/compare channels.
Parameters	• **TIMx:** Timer instance • **Channels:** This parameter can be a combination of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH1N – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH2N – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH3N – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCER CC1E LL_TIM_CC_EnableChannel • CCER CC1NE LL_TIM_CC_EnableChannel • CCER CC2E LL_TIM_CC_EnableChannel • CCER CC2NE LL_TIM_CC_EnableChannel • CCER CC3E LL_TIM_CC_EnableChannel • CCER CC3NE LL_TIM_CC_EnableChannel • CCER CC4E LL_TIM_CC_EnableChannel • CCER CC5E LL_TIM_CC_EnableChannel • CCER CC6E LL_TIM_CC_EnableChannel

LL_TIM_CC_DisableChannel

Function name	**__STATIC_INLINE void LL_TIM_CC_DisableChannel (TIM_TypeDef * TIMx, uint32_t Channels)**
Function description	Disable capture/compare channels.
Parameters	• **TIMx:** Timer instance • **Channels:** This parameter can be a combination of the

following values:
- – LL_TIM_CHANNEL_CH1
- – LL_TIM_CHANNEL_CH1N
- – LL_TIM_CHANNEL_CH2
- – LL_TIM_CHANNEL_CH2N
- – LL_TIM_CHANNEL_CH3
- – LL_TIM_CHANNEL_CH3N
- – LL_TIM_CHANNEL_CH4
- – LL_TIM_CHANNEL_CH5
- – LL_TIM_CHANNEL_CH6

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCER CC1E LL_TIM_CC_DisableChannel
- CCER CC1NE LL_TIM_CC_DisableChannel
- CCER CC2E LL_TIM_CC_DisableChannel
- CCER CC2NE LL_TIM_CC_DisableChannel
- CCER CC3E LL_TIM_CC_DisableChannel
- CCER CC3NE LL_TIM_CC_DisableChannel
- CCER CC4E LL_TIM_CC_DisableChannel
- CCER CC5E LL_TIM_CC_DisableChannel
- CCER CC6E LL_TIM_CC_DisableChannel

LL_TIM_CC_IsEnabledChannel

Function name
__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description
Indicate whether channel(s) is(are) enabled.

Parameters
- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - – LL_TIM_CHANNEL_CH1
  - – LL_TIM_CHANNEL_CH1N
  - – LL_TIM_CHANNEL_CH2
  - – LL_TIM_CHANNEL_CH2N
  - – LL_TIM_CHANNEL_CH3
  - – LL_TIM_CHANNEL_CH3N
  - – LL_TIM_CHANNEL_CH4
  - – LL_TIM_CHANNEL_CH5
  - – LL_TIM_CHANNEL_CH6

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CCER CC1E LL_TIM_CC_IsEnabledChannel
- CCER CC1NE LL_TIM_CC_IsEnabledChannel
- CCER CC2E LL_TIM_CC_IsEnabledChannel
- CCER CC2NE LL_TIM_CC_IsEnabledChannel
- CCER CC3E LL_TIM_CC_IsEnabledChannel
- CCER CC3NE LL_TIM_CC_IsEnabledChannel
- CCER CC4E LL_TIM_CC_IsEnabledChannel
- CCER CC5E LL_TIM_CC_IsEnabledChannel
- CCER CC6E LL_TIM_CC_IsEnabledChannel

LL_TIM_OC_ConfigOutput

Function name	**__STATIC_INLINE void LL_TIM_OC_ConfigOutput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)**
Function description	Configure an output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6 • **Configuration:** This parameter must be a combination of all the following values: – LL_TIM_OCPOLARITY_HIGH or LL_TIM_OCPOLARITY_LOW – LL_TIM_OCIDLESTATE_LOW or LL_TIM_OCIDLESTATE_HIGH
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCMR1 CC1S LL_TIM_OC_ConfigOutput • CCMR1 CC2S LL_TIM_OC_ConfigOutput • CCMR2 CC3S LL_TIM_OC_ConfigOutput • CCMR2 CC4S LL_TIM_OC_ConfigOutput • CCMR3 CC5S LL_TIM_OC_ConfigOutput • CCMR3 CC6S LL_TIM_OC_ConfigOutput • CCER CC1P LL_TIM_OC_ConfigOutput • CCER CC2P LL_TIM_OC_ConfigOutput • CCER CC3P LL_TIM_OC_ConfigOutput • CCER CC4P LL_TIM_OC_ConfigOutput • CCER CC5P LL_TIM_OC_ConfigOutput • CCER CC6P LL_TIM_OC_ConfigOutput • CR2 OIS1 LL_TIM_OC_ConfigOutput • CR2 OIS2 LL_TIM_OC_ConfigOutput • CR2 OIS3 LL_TIM_OC_ConfigOutput • CR2 OIS4 LL_TIM_OC_ConfigOutput • CR2 OIS5 LL_TIM_OC_ConfigOutput • CR2 OIS6 LL_TIM_OC_ConfigOutput

LL_TIM_OC_SetMode

Function name	**__STATIC_INLINE void LL_TIM_OC_SetMode (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Mode)**
Function description	Define the behavior of the output reference signal OCxREF from which OCx and OCxN (when relevant) are derived.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2

- – LL_TIM_CHANNEL_CH3
- – LL_TIM_CHANNEL_CH4
- – LL_TIM_CHANNEL_CH5
- – LL_TIM_CHANNEL_CH6
- **Mode:** This parameter can be one of the following values:
  - – LL_TIM_OCMODE_FROZEN
  - – LL_TIM_OCMODE_ACTIVE
  - – LL_TIM_OCMODE_INACTIVE
  - – LL_TIM_OCMODE_TOGGLE
  - – LL_TIM_OCMODE_FORCED_INACTIVE
  - – LL_TIM_OCMODE_FORCED_ACTIVE
  - – LL_TIM_OCMODE_PWM1
  - – LL_TIM_OCMODE_PWM2
  - – LL_TIM_OCMODE_RETRIG_OPM1
  - – LL_TIM_OCMODE_RETRIG_OPM2
  - – LL_TIM_OCMODE_COMBINED_PWM1
  - – LL_TIM_OCMODE_COMBINED_PWM2
  - – LL_TIM_OCMODE_ASSYMETRIC_PWM1
  - – LL_TIM_OCMODE_ASSYMETRIC_PWM2

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCMR1 OC1M LL_TIM_OC_SetMode
- CCMR1 OC2M LL_TIM_OC_SetMode
- CCMR2 OC3M LL_TIM_OC_SetMode
- CCMR2 OC4M LL_TIM_OC_SetMode
- CCMR3 OC5M LL_TIM_OC_SetMode
- CCMR3 OC6M LL_TIM_OC_SetMode

LL_TIM_OC_GetMode

Function name
__STATIC_INLINE uint32_t LL_TIM_OC_GetMode (TIM_TypeDef * TIMx, uint32_t Channel)

Function description
Get the output compare mode of an output channel.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - – LL_TIM_CHANNEL_CH1
  - – LL_TIM_CHANNEL_CH2
  - – LL_TIM_CHANNEL_CH3
  - – LL_TIM_CHANNEL_CH4
  - – LL_TIM_CHANNEL_CH5
  - – LL_TIM_CHANNEL_CH6

Return values
- **Returned:** value can be one of the following values:
  - – LL_TIM_OCMODE_FROZEN
  - – LL_TIM_OCMODE_ACTIVE
  - – LL_TIM_OCMODE_INACTIVE
  - – LL_TIM_OCMODE_TOGGLE
  - – LL_TIM_OCMODE_FORCED_INACTIVE
  - – LL_TIM_OCMODE_FORCED_ACTIVE
  - – LL_TIM_OCMODE_PWM1
  - – LL_TIM_OCMODE_PWM2
  - – LL_TIM_OCMODE_RETRIG_OPM1

– LL_TIM_OCMODE_RETRIG_OPM2
– LL_TIM_OCMODE_COMBINED_PWM1
– LL_TIM_OCMODE_COMBINED_PWM2
– LL_TIM_OCMODE_ASSYMETRIC_PWM1
– LL_TIM_OCMODE_ASSYMETRIC_PWM2

Reference Manual to LL API cross reference:	• CCMR1 OC1M LL_TIM_OC_GetMode • CCMR1 OC2M LL_TIM_OC_GetMode • CCMR2 OC3M LL_TIM_OC_GetMode • CCMR2 OC4M LL_TIM_OC_GetMode • CCMR3 OC5M LL_TIM_OC_GetMode • CCMR3 OC6M LL_TIM_OC_GetMode

LL_TIM_OC_SetPolarity

Function name	**__STATIC_INLINE void LL_TIM_OC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Polarity)**
Function description	Set the polarity of an output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH1N – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH2N – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH3N – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6 • **Polarity:** This parameter can be one of the following values: – LL_TIM_OCPOLARITY_HIGH – LL_TIM_OCPOLARITY_LOW
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCER CC1P LL_TIM_OC_SetPolarity • CCER CC1NP LL_TIM_OC_SetPolarity • CCER CC2P LL_TIM_OC_SetPolarity • CCER CC2NP LL_TIM_OC_SetPolarity • CCER CC3P LL_TIM_OC_SetPolarity • CCER CC3NP LL_TIM_OC_SetPolarity • CCER CC4P LL_TIM_OC_SetPolarity • CCER CC5P LL_TIM_OC_SetPolarity • CCER CC6P LL_TIM_OC_SetPolarity

LL_TIM_OC_GetPolarity

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetPolarity (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Get the polarity of an output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values:

	– LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH1N – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH2N – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH3N – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **Returned:** value can be one of the following values: – LL_TIM_OCPOLARITY_HIGH – LL_TIM_OCPOLARITY_LOW
Reference Manual to LL API cross reference:	• CCER CC1P LL_TIM_OC_GetPolarity • CCER CC1NP LL_TIM_OC_GetPolarity • CCER CC2P LL_TIM_OC_GetPolarity • CCER CC2NP LL_TIM_OC_GetPolarity • CCER CC3P LL_TIM_OC_GetPolarity • CCER CC3NP LL_TIM_OC_GetPolarity • CCER CC4P LL_TIM_OC_GetPolarity • CCER CC5P LL_TIM_OC_GetPolarity • CCER CC6P LL_TIM_OC_GetPolarity

LL_TIM_OC_SetIdleState

Function name	**__STATIC_INLINE void LL_TIM_OC_SetIdleState (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t IdleState)**
Function description	Set the IDLE state of an output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH1N – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH2N – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH3N – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6 • **IdleState:** This parameter can be one of the following values: – LL_TIM_OCIDLESTATE_LOW – LL_TIM_OCIDLESTATE_HIGH
Return values	• **None:**
Notes	• This function is significant only for the timer instances supporting the break feature. Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• CR2 OIS1 LL_TIM_OC_SetIdleState • CR2 OIS2N LL_TIM_OC_SetIdleState • CR2 OIS2 LL_TIM_OC_SetIdleState

- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS3 LL_TIM_OC_SetIdleState
- CR2 OIS3N LL_TIM_OC_SetIdleState
- CR2 OIS4 LL_TIM_OC_SetIdleState
- CR2 OIS5 LL_TIM_OC_SetIdleState
- CR2 OIS6 LL_TIM_OC_SetIdleState

LL_TIM_OC_GetIdleState

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetIdleState (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Get the IDLE state of an output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH1N – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH2N – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH3N – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **Returned:** value can be one of the following values: – LL_TIM_OCIDLESTATE_LOW – LL_TIM_OCIDLESTATE_HIGH
Reference Manual to LL API cross reference:	• CR2 OIS1 LL_TIM_OC_GetIdleState • CR2 OIS2N LL_TIM_OC_GetIdleState • CR2 OIS2 LL_TIM_OC_GetIdleState • CR2 OIS2N LL_TIM_OC_GetIdleState • CR2 OIS3 LL_TIM_OC_GetIdleState • CR2 OIS3N LL_TIM_OC_GetIdleState • CR2 OIS4 LL_TIM_OC_GetIdleState • CR2 OIS5 LL_TIM_OC_GetIdleState • CR2 OIS6 LL_TIM_OC_GetIdleState

LL_TIM_OC_EnableFast

Function name	**__STATIC_INLINE void LL_TIM_OC_EnableFast (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Enable fast mode for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6

Return values	• **None:**
Notes	• Acts only if the channel is configured in PWM1 or PWM2 mode.
Reference Manual to LL API cross reference:	• CCMR1 OC1FE LL_TIM_OC_EnableFast • CCMR1 OC2FE LL_TIM_OC_EnableFast • CCMR2 OC3FE LL_TIM_OC_EnableFast • CCMR2 OC4FE LL_TIM_OC_EnableFast • CCMR3 OC5FE LL_TIM_OC_EnableFast • CCMR3 OC6FE LL_TIM_OC_EnableFast

LL_TIM_OC_DisableFast

Function name	**__STATIC_INLINE void LL_TIM_OC_DisableFast (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Disable fast mode for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCMR1 OC1FE LL_TIM_OC_DisableFast • CCMR1 OC2FE LL_TIM_OC_DisableFast • CCMR2 OC3FE LL_TIM_OC_DisableFast • CCMR2 OC4FE LL_TIM_OC_DisableFast • CCMR3 OC5FE LL_TIM_OC_DisableFast • CCMR3 OC6FE LL_TIM_OC_DisableFast

LL_TIM_OC_IsEnabledFast

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledFast (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Indicates whether fast mode is enabled for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CCMR1 OC1FE LL_TIM_OC_IsEnabledFast • CCMR1 OC2FE LL_TIM_OC_IsEnabledFast • CCMR2 OC3FE LL_TIM_OC_IsEnabledFast

reference:	• CCMR2 OC4FE LL_TIM_OC_IsEnabledFast • CCMR3 OC5FE LL_TIM_OC_IsEnabledFast • CCMR3 OC6FE LL_TIM_OC_IsEnabledFast

LL_TIM_OC_EnablePreload

Function name	**__STATIC_INLINE void LL_TIM_OC_EnablePreload (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Enable compare register (TIMx_CCRx) preload for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCMR1 OC1PE LL_TIM_OC_EnablePreload • CCMR1 OC2PE LL_TIM_OC_EnablePreload • CCMR2 OC3PE LL_TIM_OC_EnablePreload • CCMR2 OC4PE LL_TIM_OC_EnablePreload • CCMR3 OC5PE LL_TIM_OC_EnablePreload • CCMR3 OC6PE LL_TIM_OC_EnablePreload

LL_TIM_OC_DisablePreload

Function name	**__STATIC_INLINE void LL_TIM_OC_DisablePreload (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Disable compare register (TIMx_CCRx) preload for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCMR1 OC1PE LL_TIM_OC_DisablePreload • CCMR1 OC2PE LL_TIM_OC_DisablePreload • CCMR2 OC3PE LL_TIM_OC_DisablePreload • CCMR2 OC4PE LL_TIM_OC_DisablePreload • CCMR3 OC5PE LL_TIM_OC_DisablePreload • CCMR3 OC6PE LL_TIM_OC_DisablePreload

LL_TIM_OC_IsEnabledPreload

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledPreload (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Indicates whether compare register (TIMx_CCRx) preload is enabled for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CCMR1 OC1PE LL_TIM_OC_IsEnabledPreload • CCMR1 OC2PE LL_TIM_OC_IsEnabledPreload • CCMR2 OC3PE LL_TIM_OC_IsEnabledPreload • CCMR2 OC4PE LL_TIM_OC_IsEnabledPreload • CCMR3 OC5PE LL_TIM_OC_IsEnabledPreload • CCMR3 OC6PE LL_TIM_OC_IsEnabledPreload

LL_TIM_OC_EnableClear

Function name	**__STATIC_INLINE void LL_TIM_OC_EnableClear (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Enable clearing the output channel on an external event.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **None:**
Notes	• This function can only be used in Output compare and PWM modes. It does not work in Forced mode. • Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.
Reference Manual to LL API cross reference:	• CCMR1 OC1CE LL_TIM_OC_EnableClear • CCMR1 OC2CE LL_TIM_OC_EnableClear • CCMR2 OC3CE LL_TIM_OC_EnableClear • CCMR2 OC4CE LL_TIM_OC_EnableClear • CCMR3 OC5CE LL_TIM_OC_EnableClear • CCMR3 OC6CE LL_TIM_OC_EnableClear

LL_TIM_OC_DisableClear

Function name	**__STATIC_INLINE void LL_TIM_OC_DisableClear (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Disable clearing the output channel on an external event.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **None:**
Notes	• Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.
Reference Manual to LL API cross reference:	• CCMR1 OC1CE LL_TIM_OC_DisableClear • CCMR1 OC2CE LL_TIM_OC_DisableClear • CCMR2 OC3CE LL_TIM_OC_DisableClear • CCMR2 OC4CE LL_TIM_OC_DisableClear • CCMR3 OC5CE LL_TIM_OC_DisableClear • CCMR3 OC6CE LL_TIM_OC_DisableClear

LL_TIM_OC_IsEnabledClear

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledClear (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Indicates clearing the output channel on an external event is enabled for the output channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6
Return values	• **State:** of bit (1 or 0).
Notes	• This function enables clearing the output channel on an external event. • This function can only be used in Output compare and PWM modes. It does not work in Forced mode. • Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.
Reference Manual to LL API cross	• CCMR1 OC1CE LL_TIM_OC_IsEnabledClear • CCMR1 OC2CE LL_TIM_OC_IsEnabledClear • CCMR2 OC3CE LL_TIM_OC_IsEnabledClear

reference:	• CCMR2 OC4CE LL_TIM_OC_IsEnabledClear • CCMR3 OC5CE LL_TIM_OC_IsEnabledClear • CCMR3 OC6CE LL_TIM_OC_IsEnabledClear

LL_TIM_OC_SetDeadTime

Function name	**__STATIC_INLINE void LL_TIM_OC_SetDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)**
Function description	Set the dead-time delay (delay inserted between the rising edge of the OCxREF signal and the rising edge if the Ocx and OCxN signals).
Parameters	• **TIMx:** Timer instance • **DeadTime:** between Min_Data=0 and Max_Data=255
Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not dead-time insertion feature is supported by a timer instance. • Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter
Reference Manual to LL API cross reference:	• BDTR DTG LL_TIM_OC_SetDeadTime

LL_TIM_OC_SetCompareCH1

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH1 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 1 (TIMx_CCR1).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**
Notes	• In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR1 CCR1 LL_TIM_OC_SetCompareCH1

LL_TIM_OC_SetCompareCH2

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH2 (TIM_TypeDef * TIMx, uint32_t CompareValue)**

Function description	Set compare value for output channel 2 (TIMx_CCR2).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**
Notes	• In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR2 CCR2 LL_TIM_OC_SetCompareCH2

LL_TIM_OC_SetCompareCH3

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH3 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 3 (TIMx_CCR3).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**
Notes	• In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR3 CCR3 LL_TIM_OC_SetCompareCH3

LL_TIM_OC_SetCompareCH4

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH4 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 4 (TIMx_CCR4).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**

Notes	• In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR4 CCR4 LL_TIM_OC_SetCompareCH4

LL_TIM_OC_SetCompareCH5

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH5 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 5 (TIMx_CCR5).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**
Notes	• Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR5 CCR5 LL_TIM_OC_SetCompareCH5

LL_TIM_OC_SetCompareCH6

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH6 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 6 (TIMx_CCR6).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**
Notes	• Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR6 CCR6 LL_TIM_OC_SetCompareCH6

LL_TIM_OC_GetCompareCH1

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH1 (TIM_TypeDef * TIMx)**

Function description	Get compare value (TIMx_CCR1) set for output channel 1.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR1 CCR1 LL_TIM_OC_GetCompareCH1

LL_TIM_OC_GetCompareCH2

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH2 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR2) set for output channel 2.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR2 CCR2 LL_TIM_OC_GetCompareCH2

LL_TIM_OC_GetCompareCH3

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH3 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR3) set for output channel 3.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32

	bits counter. • Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel 3 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR3 CCR3 LL_TIM_OC_GetCompareCH3

LL_TIM_OC_GetCompareCH4

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH4 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR4) set for output channel 4.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR4 CCR4 LL_TIM_OC_GetCompareCH4

LL_TIM_OC_GetCompareCH5

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH5 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR5) set for output channel 5.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR5 CCR5 LL_TIM_OC_GetCompareCH5

LL_TIM_OC_GetCompareCH6

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH6 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR6) set for output channel 6.

Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR6 CCR6 LL_TIM_OC_GetCompareCH6

LL_TIM_SetCH5CombinedChannels

Function name	**__STATIC_INLINE void LL_TIM_SetCH5CombinedChannels (TIM_TypeDef * TIMx, uint32_t GroupCH5)**
Function description	Select on which reference signal the OC5REF is combined to.
Parameters	• **TIMx:** Timer instance • **GroupCH5:** This parameter can be one of the following values: – LL_TIM_GROUPCH5_NONE – LL_TIM_GROUPCH5_OC1REFC – LL_TIM_GROUPCH5_OC2REFC – LL_TIM_GROUPCH5_OC3REFC
Return values	• **None:**
Notes	• Macro IS_TIM_COMBINED3PHASEPWM_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the combined 3-phase PWM mode.
Reference Manual to LL API cross reference:	• CCR5 GC5C3 LL_TIM_SetCH5CombinedChannels • CCR5 GC5C2 LL_TIM_SetCH5CombinedChannels • CCR5 GC5C1 LL_TIM_SetCH5CombinedChannels

LL_TIM_IC_Config

Function name	**__STATIC_INLINE void LL_TIM_IC_Config (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)**
Function description	Configure input channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **Configuration:** This parameter must be a combination of all the following values: – LL_TIM_ACTIVEINPUT_DIRECTTI or LL_TIM_ACTIVEINPUT_INDIRECTTI or LL_TIM_ACTIVEINPUT_TRC – LL_TIM_ICPSC_DIV1 or ... or LL_TIM_ICPSC_DIV8 – LL_TIM_IC_FILTER_FDIV1 or ... or

- LL_TIM_IC_FILTER_FDIV32_N8
- – LL_TIM_IC_POLARITY_RISING or LL_TIM_IC_POLARITY_FALLING or LL_TIM_IC_POLARITY_BOTHEDGE

Return values	• **None:**
Reference Manual to LL API cross reference:	• CCMR1 CC1S LL_TIM_IC_Config • CCMR1 IC1PSC LL_TIM_IC_Config • CCMR1 IC1F LL_TIM_IC_Config • CCMR1 CC2S LL_TIM_IC_Config • CCMR1 IC2PSC LL_TIM_IC_Config • CCMR1 IC2F LL_TIM_IC_Config • CCMR2 CC3S LL_TIM_IC_Config • CCMR2 IC3PSC LL_TIM_IC_Config • CCMR2 IC3F LL_TIM_IC_Config • CCMR2 CC4S LL_TIM_IC_Config • CCMR2 IC4PSC LL_TIM_IC_Config • CCMR2 IC4F LL_TIM_IC_Config • CCER CC1P LL_TIM_IC_Config • CCER CC1NP LL_TIM_IC_Config • CCER CC2P LL_TIM_IC_Config • CCER CC2NP LL_TIM_IC_Config • CCER CC3P LL_TIM_IC_Config • CCER CC3NP LL_TIM_IC_Config • CCER CC4P LL_TIM_IC_Config • CCER CC4NP LL_TIM_IC_Config

LL_TIM_IC_SetActiveInput

Function name	**__STATIC_INLINE void LL_TIM_IC_SetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICActiveInput)**
Function description	Set the active input.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **ICActiveInput:** This parameter can be one of the following values: – LL_TIM_ACTIVEINPUT_DIRECTTI – LL_TIM_ACTIVEINPUT_INDIRECTTI – LL_TIM_ACTIVEINPUT_TRC
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCMR1 CC1S LL_TIM_IC_SetActiveInput • CCMR1 CC2S LL_TIM_IC_SetActiveInput • CCMR2 CC3S LL_TIM_IC_SetActiveInput • CCMR2 CC4S LL_TIM_IC_SetActiveInput

LL_TIM_IC_GetActiveInput

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Get the current active input.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **Returned:** value can be one of the following values: – LL_TIM_ACTIVEINPUT_DIRECTTI – LL_TIM_ACTIVEINPUT_INDIRECTTI – LL_TIM_ACTIVEINPUT_TRC
Reference Manual to LL API cross reference:	• CCMR1 CC1S LL_TIM_IC_GetActiveInput • CCMR1 CC2S LL_TIM_IC_GetActiveInput • CCMR2 CC3S LL_TIM_IC_GetActiveInput • CCMR2 CC4S LL_TIM_IC_GetActiveInput

LL_TIM_IC_SetPrescaler

Function name	**__STATIC_INLINE void LL_TIM_IC_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPrescaler)**
Function description	Set the prescaler of input channel.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **ICPrescaler:** This parameter can be one of the following values: – LL_TIM_ICPSC_DIV1 – LL_TIM_ICPSC_DIV2 – LL_TIM_ICPSC_DIV4 – LL_TIM_ICPSC_DIV8
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCMR1 IC1PSC LL_TIM_IC_SetPrescaler • CCMR1 IC2PSC LL_TIM_IC_SetPrescaler • CCMR2 IC3PSC LL_TIM_IC_SetPrescaler • CCMR2 IC4PSC LL_TIM_IC_SetPrescaler

LL_TIM_IC_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Get the current prescaler value acting on an input channel.

Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **Returned:** value can be one of the following values: – LL_TIM_ICPSC_DIV1 – LL_TIM_ICPSC_DIV2 – LL_TIM_ICPSC_DIV4 – LL_TIM_ICPSC_DIV8
Reference Manual to LL API cross reference:	• CCMR1 IC1PSC LL_TIM_IC_GetPrescaler • CCMR1 IC2PSC LL_TIM_IC_GetPrescaler • CCMR2 IC3PSC LL_TIM_IC_GetPrescaler • CCMR2 IC4PSC LL_TIM_IC_GetPrescaler

LL_TIM_IC_SetFilter

Function name	**__STATIC_INLINE void LL_TIM_IC_SetFilter (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICFilter)**
Function description	Set the input filter duration.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **ICFilter:** This parameter can be one of the following values: – LL_TIM_IC_FILTER_FDIV1 – LL_TIM_IC_FILTER_FDIV1_N2 – LL_TIM_IC_FILTER_FDIV1_N4 – LL_TIM_IC_FILTER_FDIV1_N8 – LL_TIM_IC_FILTER_FDIV2_N6 – LL_TIM_IC_FILTER_FDIV2_N8 – LL_TIM_IC_FILTER_FDIV4_N6 – LL_TIM_IC_FILTER_FDIV4_N8 – LL_TIM_IC_FILTER_FDIV8_N6 – LL_TIM_IC_FILTER_FDIV8_N8 – LL_TIM_IC_FILTER_FDIV16_N5 – LL_TIM_IC_FILTER_FDIV16_N6 – LL_TIM_IC_FILTER_FDIV16_N8 – LL_TIM_IC_FILTER_FDIV32_N5 – LL_TIM_IC_FILTER_FDIV32_N6 – LL_TIM_IC_FILTER_FDIV32_N8
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCMR1 IC1F LL_TIM_IC_SetFilter • CCMR1 IC2F LL_TIM_IC_SetFilter • CCMR2 IC3F LL_TIM_IC_SetFilter • CCMR2 IC4F LL_TIM_IC_SetFilter

LL_TIM_IC_GetFilter

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetFilter (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Get the input filter duration.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **Returned:** value can be one of the following values: – LL_TIM_IC_FILTER_FDIV1 – LL_TIM_IC_FILTER_FDIV1_N2 – LL_TIM_IC_FILTER_FDIV1_N4 – LL_TIM_IC_FILTER_FDIV1_N8 – LL_TIM_IC_FILTER_FDIV2_N6 – LL_TIM_IC_FILTER_FDIV2_N8 – LL_TIM_IC_FILTER_FDIV4_N6 – LL_TIM_IC_FILTER_FDIV4_N8 – LL_TIM_IC_FILTER_FDIV8_N6 – LL_TIM_IC_FILTER_FDIV8_N8 – LL_TIM_IC_FILTER_FDIV16_N5 – LL_TIM_IC_FILTER_FDIV16_N6 – LL_TIM_IC_FILTER_FDIV16_N8 – LL_TIM_IC_FILTER_FDIV32_N5 – LL_TIM_IC_FILTER_FDIV32_N6 – LL_TIM_IC_FILTER_FDIV32_N8
Reference Manual to LL API cross reference:	• CCMR1 IC1F LL_TIM_IC_GetFilter • CCMR1 IC2F LL_TIM_IC_GetFilter • CCMR2 IC3F LL_TIM_IC_GetFilter • CCMR2 IC4F LL_TIM_IC_GetFilter

LL_TIM_IC_SetPolarity

Function name	**__STATIC_INLINE void LL_TIM_IC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPolarity)**
Function description	Set the input channel polarity.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **ICPolarity:** This parameter can be one of the following values: – LL_TIM_IC_POLARITY_RISING – LL_TIM_IC_POLARITY_FALLING – LL_TIM_IC_POLARITY_BOTHEDGE

Return values	• **None:**
Reference Manual to LL API cross reference:	• CCER CC1P LL_TIM_IC_SetPolarity • CCER CC1NP LL_TIM_IC_SetPolarity • CCER CC2P LL_TIM_IC_SetPolarity • CCER CC2NP LL_TIM_IC_SetPolarity • CCER CC3P LL_TIM_IC_SetPolarity • CCER CC3NP LL_TIM_IC_SetPolarity • CCER CC4P LL_TIM_IC_SetPolarity • CCER CC4NP LL_TIM_IC_SetPolarity

LL_TIM_IC_GetPolarity

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetPolarity (TIM_TypeDef * TIMx, uint32_t Channel)**
Function description	Get the current input channel polarity.
Parameters	• **TIMx:** Timer instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4
Return values	• **Returned:** value can be one of the following values: – LL_TIM_IC_POLARITY_RISING – LL_TIM_IC_POLARITY_FALLING – LL_TIM_IC_POLARITY_BOTHEDGE
Reference Manual to LL API cross reference:	• CCER CC1P LL_TIM_IC_GetPolarity • CCER CC1NP LL_TIM_IC_GetPolarity • CCER CC2P LL_TIM_IC_GetPolarity • CCER CC2NP LL_TIM_IC_GetPolarity • CCER CC3P LL_TIM_IC_GetPolarity • CCER CC3NP LL_TIM_IC_GetPolarity • CCER CC4P LL_TIM_IC_GetPolarity • CCER CC4NP LL_TIM_IC_GetPolarity

LL_TIM_IC_EnableXORCombination

Function name	**__STATIC_INLINE void LL_TIM_IC_EnableXORCombination (TIM_TypeDef * TIMx)**
Function description	Connect the TIMx_CH1, CH2 and CH3 pins to the TI1 input (XOR combination).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.
Reference Manual to LL API cross reference:	• CR2 TI1S LL_TIM_IC_EnableXORCombination

LL_TIM_IC_DisableXORCombination

Function name	**__STATIC_INLINE void LL_TIM_IC_DisableXORCombination (TIM_TypeDef * TIMx)**
Function description	Disconnect the TIMx_CH1, CH2 and CH3 pins from the TI1 input.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.
Reference Manual to LL API cross reference:	• CR2 TI1S LL_TIM_IC_DisableXORCombination

LL_TIM_IC_IsEnabledXORCombination

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_IsEnabledXORCombination (TIM_TypeDef * TIMx)**
Function description	Indicates whether the TIMx_CH1, CH2 and CH3 pins are connectected to the TI1 input.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.
Reference Manual to LL API cross reference:	• CR2 TI1S LL_TIM_IC_IsEnabledXORCombination

LL_TIM_IC_GetCaptureCH1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 1.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not input channel 1 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR1 CCR1 LL_TIM_IC_GetCaptureCH1

LL_TIM_IC_GetCaptureCH2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH2 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 2.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not input channel 2 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR2 CCR2 LL_TIM_IC_GetCaptureCH2

LL_TIM_IC_GetCaptureCH3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH3 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 3.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not input channel 3 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR3 CCR3 LL_TIM_IC_GetCaptureCH3

LL_TIM_IC_GetCaptureCH4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH4 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 4.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes	• In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF. • Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter. • Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not input channel 4 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR4 CCR4 LL_TIM_IC_GetCaptureCH4

LL_TIM_EnableExternalClock

Function name	**__STATIC_INLINE void LL_TIM_EnableExternalClock (TIM_TypeDef * TIMx)**
Function description	Enable external clock mode 2.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• When external clock mode 2 is enabled the counter is clocked by any active edge on the ETRF signal. • Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR ECE LL_TIM_EnableExternalClock

LL_TIM_DisableExternalClock

Function name	**__STATIC_INLINE void LL_TIM_DisableExternalClock (TIM_TypeDef * TIMx)**
Function description	Disable external clock mode 2.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR ECE LL_TIM_DisableExternalClock

LL_TIM_IsEnabledExternalClock

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledExternalClock (TIM_TypeDef * TIMx)**

Function description	Indicate whether external clock mode 2 is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR ECE LL_TIM_IsEnabledExternalClock

LL_TIM_SetClockSource

Function name	**__STATIC_INLINE void LL_TIM_SetClockSource (TIM_TypeDef * TIMx, uint32_t ClockSource)**
Function description	Set the clock source of the counter clock.
Parameters	• **TIMx:** Timer instance • **ClockSource:** This parameter can be one of the following values: – LL_TIM_CLOCKSOURCE_INTERNAL – LL_TIM_CLOCKSOURCE_EXT_MODE1 – LL_TIM_CLOCKSOURCE_EXT_MODE2
Return values	• **None:**
Notes	• when selected clock source is external clock mode 1, the timer input the external clock is applied is selected by calling the LL_TIM_SetTriggerInput() function. This timer input must be configured by calling the LL_TIM_IC_Config() function. • Macro IS_TIM_CLOCKSOURCE_ETRMODE1_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode1. • Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR SMS LL_TIM_SetClockSource • SMCR ECE LL_TIM_SetClockSource

LL_TIM_SetEncoderMode

Function name	**__STATIC_INLINE void LL_TIM_SetEncoderMode (TIM_TypeDef * TIMx, uint32_t EncoderMode)**
Function description	Set the encoder interface mode.
Parameters	• **TIMx:** Timer instance • **EncoderMode:** This parameter can be one of the following values: – LL_TIM_ENCODERMODE_X2_TI1

	– LL_TIM_ENCODERMODE_X2_TI2 – LL_TIM_ENCODERMODE_X4_TI12
Return values	• **None:**
Notes	• Macro IS_TIM_ENCODER_INTERFACE_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the encoder mode.
Reference Manual to LL API cross reference:	• SMCR SMS LL_TIM_SetEncoderMode

LL_TIM_SetTriggerOutput

Function name	**__STATIC_INLINE void LL_TIM_SetTriggerOutput (TIM_TypeDef * TIMx, uint32_t TimerSynchronization)**
Function description	Set the trigger output (TRGO) used for timer synchronization .
Parameters	• **TIMx:** Timer instance • **TimerSynchronization:** This parameter can be one of the following values: – LL_TIM_TRGO_RESET – LL_TIM_TRGO_ENABLE – LL_TIM_TRGO_UPDATE – LL_TIM_TRGO_CC1IF – LL_TIM_TRGO_OC1REF – LL_TIM_TRGO_OC2REF – LL_TIM_TRGO_OC3REF – LL_TIM_TRGO_OC4REF
Return values	• **None:**
Notes	• Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a master timer.
Reference Manual to LL API cross reference:	• CR2 MMS LL_TIM_SetTriggerOutput

LL_TIM_SetTriggerOutput2

Function name	**__STATIC_INLINE void LL_TIM_SetTriggerOutput2 (TIM_TypeDef * TIMx, uint32_t ADCSynchronization)**
Function description	Set the trigger output 2 (TRGO2) used for ADC synchronization .
Parameters	• **TIMx:** Timer Instance • **ADCSynchronization:** This parameter can be one of the following values: – LL_TIM_TRGO2_RESET – LL_TIM_TRGO2_ENABLE – LL_TIM_TRGO2_UPDATE – LL_TIM_TRGO2_CC1F – LL_TIM_TRGO2_OC1 – LL_TIM_TRGO2_OC2

– LL_TIM_TRGO2_OC3
– LL_TIM_TRGO2_OC4
– LL_TIM_TRGO2_OC5
– LL_TIM_TRGO2_OC6
– LL_TIM_TRGO2_OC4_RISINGFALLING
– LL_TIM_TRGO2_OC6_RISINGFALLING
– LL_TIM_TRGO2_OC4_RISING_OC6_RISING
– LL_TIM_TRGO2_OC4_RISING_OC6_FALLING
– LL_TIM_TRGO2_OC5_RISING_OC6_RISING
– LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

Return values
- **None:**

Notes
- Macro IS_TIM_TRGO2_INSTANCE(TIMx) can be used to check whether or not a timer instance can be used for ADC synchronization.

Reference Manual to LL API cross reference:
- CR2 MMS2 LL_TIM_SetTriggerOutput2

LL_TIM_SetSlaveMode

Function name
: **__STATIC_INLINE void LL_TIM_SetSlaveMode (TIM_TypeDef * TIMx, uint32_t SlaveMode)**

Function description
: Set the synchronization mode of a slave timer.

Parameters
- **TIMx:** Timer instance
- **SlaveMode:** This parameter can be one of the following values:
  – LL_TIM_SLAVEMODE_DISABLED
  – LL_TIM_SLAVEMODE_RESET
  – LL_TIM_SLAVEMODE_GATED
  – LL_TIM_SLAVEMODE_TRIGGER
  – LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Return values
- **None:**

Notes
- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:
- SMCR SMS LL_TIM_SetSlaveMode

LL_TIM_SetTriggerInput

Function name
: **__STATIC_INLINE void LL_TIM_SetTriggerInput (TIM_TypeDef * TIMx, uint32_t TriggerInput)**

Function description
: Set the selects the trigger input to be used to synchronize the counter.

Parameters
- **TIMx:** Timer instance
- **TriggerInput:** This parameter can be one of the following values:
  – LL_TIM_TS_ITR0

- – LL_TIM_TS_ITR1
- – LL_TIM_TS_ITR2
- – LL_TIM_TS_ITR3
- – LL_TIM_TS_TI1F_ED
- – LL_TIM_TS_TI1FP1
- – LL_TIM_TS_TI2FP2
- – LL_TIM_TS_ETRF

Return values	• **None:**
Notes	• Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.
Reference Manual to LL API cross reference:	• SMCR TS LL_TIM_SetTriggerInput

LL_TIM_EnableMasterSlaveMode

Function name	**__STATIC_INLINE void LL_TIM_EnableMasterSlaveMode (TIM_TypeDef * TIMx)**
Function description	Enable the Master/Slave mode.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.
Reference Manual to LL API cross reference:	• SMCR MSM LL_TIM_EnableMasterSlaveMode

LL_TIM_DisableMasterSlaveMode

Function name	**__STATIC_INLINE void LL_TIM_DisableMasterSlaveMode (TIM_TypeDef * TIMx)**
Function description	Disable the Master/Slave mode.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.
Reference Manual to LL API cross reference:	• SMCR MSM LL_TIM_DisableMasterSlaveMode

LL_TIM_IsEnabledMasterSlaveMode

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledMasterSlaveMode (TIM_TypeDef * TIMx)**

Function description	Indicates whether the Master/Slave mode is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.
Reference Manual to LL API cross reference:	• SMCR MSM LL_TIM_IsEnabledMasterSlaveMode

LL_TIM_ConfigETR

Function name	**__STATIC_INLINE void LL_TIM_ConfigETR (TIM_TypeDef * TIMx, uint32_t ETRPolarity, uint32_t ETRPrescaler, uint32_t ETRFilter)**
Function description	Configure the external trigger (ETR) input.
Parameters	• **TIMx:** Timer instance • **ETRPolarity:** This parameter can be one of the following values: – LL_TIM_ETR_POLARITY_NONINVERTED – LL_TIM_ETR_POLARITY_INVERTED • **ETRPrescaler:** This parameter can be one of the following values: – LL_TIM_ETR_PRESCALER_DIV1 – LL_TIM_ETR_PRESCALER_DIV2 – LL_TIM_ETR_PRESCALER_DIV4 – LL_TIM_ETR_PRESCALER_DIV8 • **ETRFilter:** This parameter can be one of the following values: – LL_TIM_ETR_FILTER_FDIV1 – LL_TIM_ETR_FILTER_FDIV1_N2 – LL_TIM_ETR_FILTER_FDIV1_N4 – LL_TIM_ETR_FILTER_FDIV1_N8 – LL_TIM_ETR_FILTER_FDIV2_N6 – LL_TIM_ETR_FILTER_FDIV2_N8 – LL_TIM_ETR_FILTER_FDIV4_N6 – LL_TIM_ETR_FILTER_FDIV4_N8 – LL_TIM_ETR_FILTER_FDIV8_N6 – LL_TIM_ETR_FILTER_FDIV8_N8 – LL_TIM_ETR_FILTER_FDIV16_N5 – LL_TIM_ETR_FILTER_FDIV16_N6 – LL_TIM_ETR_FILTER_FDIV16_N8 – LL_TIM_ETR_FILTER_FDIV32_N5 – LL_TIM_ETR_FILTER_FDIV32_N6 – LL_TIM_ETR_FILTER_FDIV32_N8
Return values	• **None:**
Notes	• Macro IS_TIM_ETR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an external trigger input.

Reference Manual to LL API cross reference:	• SMCR ETP LL_TIM_ConfigETR • SMCR ETPS LL_TIM_ConfigETR • SMCR ETF LL_TIM_ConfigETR

LL_TIM_SetETRSource

Function name	**__STATIC_INLINE void LL_TIM_SetETRSource (TIM_TypeDef * TIMx, uint32_t ETRSource)**
Function description	Select the external trigger (ETR) input source.
Parameters	• **TIMx:** Timer instance • **ETRSource:** This parameter can be one of the following values: – LL_TIM_ETRSOURCE_LEGACY – LL_TIM_ETRSOURCE_COMP1 – LL_TIM_ETRSOURCE_COMP2
Return values	• **None:**
Notes	• Macro IS_TIM_ETRSEL_INSTANCE(TIMx) can be used to check whether or not a timer instance supports ETR source selection.
Reference Manual to LL API cross reference:	• OR2 ETRSEL LL_TIM_SetETRSource

LL_TIM_EnableBRK

Function name	**__STATIC_INLINE void LL_TIM_EnableBRK (TIM_TypeDef * TIMx)**
Function description	Enable the break function.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR BKE LL_TIM_EnableBRK

LL_TIM_DisableBRK

Function name	**__STATIC_INLINE void LL_TIM_DisableBRK (TIM_TypeDef * TIMx)**
Function description	Disable the break function.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to	• BDTR BKE LL_TIM_DisableBRK

LL API cross reference:

LL_TIM_ConfigBRK

Function name	**__STATIC_INLINE void LL_TIM_ConfigBRK (TIM_TypeDef * TIMx, uint32_t BreakPolarity, uint32_t BreakFilter)**
Function description	Configure the break input.
Parameters	• **TIMx:** Timer instance • **BreakPolarity:** This parameter can be one of the following values: – LL_TIM_BREAK_POLARITY_LOW – LL_TIM_BREAK_POLARITY_HIGH • **BreakFilter:** This parameter can be one of the following values: – LL_TIM_BREAK_FILTER_FDIV1 – LL_TIM_BREAK_FILTER_FDIV1_N2 – LL_TIM_BREAK_FILTER_FDIV1_N4 – LL_TIM_BREAK_FILTER_FDIV1_N8 – LL_TIM_BREAK_FILTER_FDIV2_N6 – LL_TIM_BREAK_FILTER_FDIV2_N8 – LL_TIM_BREAK_FILTER_FDIV4_N6 – LL_TIM_BREAK_FILTER_FDIV4_N8 – LL_TIM_BREAK_FILTER_FDIV8_N6 – LL_TIM_BREAK_FILTER_FDIV8_N8 – LL_TIM_BREAK_FILTER_FDIV16_N5 – LL_TIM_BREAK_FILTER_FDIV16_N6 – LL_TIM_BREAK_FILTER_FDIV16_N8 – LL_TIM_BREAK_FILTER_FDIV32_N5 – LL_TIM_BREAK_FILTER_FDIV32_N6 – LL_TIM_BREAK_FILTER_FDIV32_N8
Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR BKP LL_TIM_ConfigBRK • BDTR BKF LL_TIM_ConfigBRK

LL_TIM_EnableBRK2

Function name	**__STATIC_INLINE void LL_TIM_EnableBRK2 (TIM_TypeDef * TIMx)**
Function description	Enable the break 2 function.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:	• BDTR BK2E LL_TIM_EnableBRK2

LL_TIM_DisableBRK2

Function name	**__STATIC_INLINE void LL_TIM_DisableBRK2 (TIM_TypeDef * TIMx)**
Function description	Disable the break 2 function.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.
Reference Manual to LL API cross reference:	• BDTR BK2E LL_TIM_DisableBRK2

LL_TIM_ConfigBRK2

Function name	**__STATIC_INLINE void LL_TIM_ConfigBRK2 (TIM_TypeDef * TIMx, uint32_t Break2Polarity, uint32_t Break2Filter)**
Function description	Configure the break 2 input.
Parameters	• **TIMx:** Timer instance • **Break2Polarity:** This parameter can be one of the following values: – LL_TIM_BREAK2_POLARITY_LOW – LL_TIM_BREAK2_POLARITY_HIGH • **Break2Filter:** This parameter can be one of the following values: – LL_TIM_BREAK2_FILTER_FDIV1 – LL_TIM_BREAK2_FILTER_FDIV1_N2 – LL_TIM_BREAK2_FILTER_FDIV1_N4 – LL_TIM_BREAK2_FILTER_FDIV1_N8 – LL_TIM_BREAK2_FILTER_FDIV2_N6 – LL_TIM_BREAK2_FILTER_FDIV2_N8 – LL_TIM_BREAK2_FILTER_FDIV4_N6 – LL_TIM_BREAK2_FILTER_FDIV4_N8 – LL_TIM_BREAK2_FILTER_FDIV8_N6 – LL_TIM_BREAK2_FILTER_FDIV8_N8 – LL_TIM_BREAK2_FILTER_FDIV16_N5 – LL_TIM_BREAK2_FILTER_FDIV16_N6 – LL_TIM_BREAK2_FILTER_FDIV16_N8 – LL_TIM_BREAK2_FILTER_FDIV32_N5 – LL_TIM_BREAK2_FILTER_FDIV32_N6 – LL_TIM_BREAK2_FILTER_FDIV32_N8
Return values	• **None:**
Notes	• Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second

break input.

Reference Manual to LL API cross reference:	• BDTR BK2P LL_TIM_ConfigBRK2 • BDTR BK2F LL_TIM_ConfigBRK2

LL_TIM_SetOffStates

Function name	**__STATIC_INLINE void LL_TIM_SetOffStates (TIM_TypeDef * TIMx, uint32_t OffStateIdle, uint32_t OffStateRun)**
Function description	Select the outputs off state (enabled v.s.
Parameters	• **TIMx:** Timer instance • **OffStateIdle:** This parameter can be one of the following values: – LL_TIM_OSSI_DISABLE – LL_TIM_OSSI_ENABLE • **OffStateRun:** This parameter can be one of the following values: – LL_TIM_OSSR_DISABLE – LL_TIM_OSSR_ENABLE
Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR OSSI LL_TIM_SetOffStates • BDTR OSSR LL_TIM_SetOffStates

LL_TIM_EnableAutomaticOutput

Function name	**__STATIC_INLINE void LL_TIM_EnableAutomaticOutput (TIM_TypeDef * TIMx)**
Function description	Enable automatic output (MOE can be set by software or automatically when a break input is active).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR AOE LL_TIM_EnableAutomaticOutput

LL_TIM_DisableAutomaticOutput

Function name	**__STATIC_INLINE void LL_TIM_DisableAutomaticOutput (TIM_TypeDef * TIMx)**
Function description	Disable automatic output (MOE can be set only by software).
Parameters	• **TIMx:** Timer instance

Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR AOE LL_TIM_DisableAutomaticOutput

LL_TIM_IsEnabledAutomaticOutput

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledAutomaticOutput (TIM_TypeDef * TIMx)**
Function description	Indicate whether automatic output is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR AOE LL_TIM_IsEnabledAutomaticOutput

LL_TIM_EnableAllOutputs

Function name	**__STATIC_INLINE void LL_TIM_EnableAllOutputs (TIM_TypeDef * TIMx)**
Function description	Enable the outputs (set the MOE bit in TIMx_BDTR register).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event • Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR MOE LL_TIM_EnableAllOutputs

LL_TIM_DisableAllOutputs

Function name	**__STATIC_INLINE void LL_TIM_DisableAllOutputs (TIM_TypeDef * TIMx)**
Function description	Disable the outputs (reset the MOE bit in TIMx_BDTR register).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event.

	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR MOE LL_TIM_DisableAllOutputs

LL_TIM_IsEnabledAllOutputs

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledAllOutputs (TIM_TypeDef * TIMx)**
Function description	Indicates whether outputs are enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR MOE LL_TIM_IsEnabledAllOutputs

LL_TIM_EnableBreakInputSource

Function name	**__STATIC_INLINE void LL_TIM_EnableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)**
Function description	Enable the signals connected to the designated timer break input.
Parameters	• **TIMx:** Timer instance • **BreakInput:** This parameter can be one of the following values: – LL_TIM_BREAK_INPUT_BKIN – LL_TIM_BREAK_INPUT_BKIN2 • **Source:** This parameter can be one of the following values: – LL_TIM_BKIN_SOURCE_BKIN – LL_TIM_BKIN_SOURCE_BKCOMP1 – LL_TIM_BKIN_SOURCE_BKCOMP2 – LL_TIM_BKIN_SOURCE_DF1BK
Return values	• **None:**
Notes	• Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.
Reference Manual to LL API cross reference:	• OR2 BKINE LL_TIM_EnableBreakInputSource • OR2 BKCMP1E LL_TIM_EnableBreakInputSource • OR2 BKCMP2E LL_TIM_EnableBreakInputSource • OR2 BKDFBK0E LL_TIM_EnableBreakInputSource • OR3 BKINE LL_TIM_EnableBreakInputSource • OR3 BKCMP1E LL_TIM_EnableBreakInputSource • OR3 BKCMP2E LL_TIM_EnableBreakInputSource • OR3 BKDFBK0E LL_TIM_EnableBreakInputSource

LL_TIM_DisableBreakInputSource

Function name	**__STATIC_INLINE void LL_TIM_DisableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)**
Function description	Disable the signals connected to the designated timer break input.
Parameters	• **TIMx:** Timer instance • **BreakInput:** This parameter can be one of the following values: – LL_TIM_BREAK_INPUT_BKIN – LL_TIM_BREAK_INPUT_BKIN2 • **Source:** This parameter can be one of the following values: – LL_TIM_BKIN_SOURCE_BKIN – LL_TIM_BKIN_SOURCE_BKCOMP1 – LL_TIM_BKIN_SOURCE_BKCOMP2 – LL_TIM_BKIN_SOURCE_DF1BK
Return values	• **None:**
Notes	• Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.
Reference Manual to LL API cross reference:	• OR2 BKINE LL_TIM_DisableBreakInputSource • OR2 BKCMP1E LL_TIM_DisableBreakInputSource • OR2 BKCMP2E LL_TIM_DisableBreakInputSource • OR2 BKDFBK0E LL_TIM_DisableBreakInputSource • OR3 BKINE LL_TIM_DisableBreakInputSource • OR3 BKCMP1E LL_TIM_DisableBreakInputSource • OR3 BKCMP2E LL_TIM_DisableBreakInputSource • OR3 BKDFBK0E LL_TIM_DisableBreakInputSource

LL_TIM_SetBreakInputSourcePolarity

Function name	**__STATIC_INLINE void LL_TIM_SetBreakInputSourcePolarity (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source, uint32_t Polarity)**
Function description	Set the polarity of the break signal for the timer break input.
Parameters	• **TIMx:** Timer instance • **BreakInput:** This parameter can be one of the following values: – LL_TIM_BREAK_INPUT_BKIN – LL_TIM_BREAK_INPUT_BKIN2 • **Source:** This parameter can be one of the following values: – LL_TIM_BKIN_SOURCE_BKIN – LL_TIM_BKIN_SOURCE_BKCOMP1 – LL_TIM_BKIN_SOURCE_BKCOMP2 • **Polarity:** This parameter can be one of the following values: – LL_TIM_BKIN_POLARITY_LOW – LL_TIM_BKIN_POLARITY_HIGH
Return values	• **None:**
Notes	• Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break

input selection.

Reference Manual to LL API cross reference:	• OR2 BKINE LL_TIM_SetBreakInputSourcePolarity • OR2 BKCMP1E LL_TIM_SetBreakInputSourcePolarity • OR2 BKCMP2E LL_TIM_SetBreakInputSourcePolarity • OR2 BKINP LL_TIM_SetBreakInputSourcePolarity • OR3 BKINE LL_TIM_SetBreakInputSourcePolarity • OR3 BKCMP1E LL_TIM_SetBreakInputSourcePolarity • OR3 BKCMP2E LL_TIM_SetBreakInputSourcePolarity • OR3 BKINP LL_TIM_SetBreakInputSourcePolarity

LL_TIM_ConfigDMABurst

Function name	**__STATIC_INLINE void LL_TIM_ConfigDMABurst (TIM_TypeDef * TIMx, uint32_t DMABurstBaseAddress, uint32_t DMABurstLength)**
Function description	Configures the timer DMA burst feature.
Parameters	• **TIMx:** Timer instance • **DMABurstBaseAddress:** This parameter can be one of the following values: – LL_TIM_DMABURST_BASEADDR_CR1 – LL_TIM_DMABURST_BASEADDR_CR2 – LL_TIM_DMABURST_BASEADDR_SMCR – LL_TIM_DMABURST_BASEADDR_DIER – LL_TIM_DMABURST_BASEADDR_SR – LL_TIM_DMABURST_BASEADDR_EGR – LL_TIM_DMABURST_BASEADDR_CCMR1 – LL_TIM_DMABURST_BASEADDR_CCMR2 – LL_TIM_DMABURST_BASEADDR_CCER – LL_TIM_DMABURST_BASEADDR_CNT – LL_TIM_DMABURST_BASEADDR_PSC – LL_TIM_DMABURST_BASEADDR_ARR – LL_TIM_DMABURST_BASEADDR_RCR – LL_TIM_DMABURST_BASEADDR_CCR1 – LL_TIM_DMABURST_BASEADDR_CCR2 – LL_TIM_DMABURST_BASEADDR_CCR3 – LL_TIM_DMABURST_BASEADDR_CCR4 – LL_TIM_DMABURST_BASEADDR_BDTR – LL_TIM_DMABURST_BASEADDR_CCMR3 – LL_TIM_DMABURST_BASEADDR_CCR5 – LL_TIM_DMABURST_BASEADDR_CCR6 – LL_TIM_DMABURST_BASEADDR_OR1 – LL_TIM_DMABURST_BASEADDR_OR2 – LL_TIM_DMABURST_BASEADDR_OR3 • **DMABurstLength:** This parameter can be one of the following values: – LL_TIM_DMABURST_LENGTH_1TRANSFER – LL_TIM_DMABURST_LENGTH_2TRANSFERS – LL_TIM_DMABURST_LENGTH_3TRANSFERS – LL_TIM_DMABURST_LENGTH_4TRANSFERS – LL_TIM_DMABURST_LENGTH_5TRANSFERS – LL_TIM_DMABURST_LENGTH_6TRANSFERS

	– LL_TIM_DMABURST_LENGTH_7TRANSFERS – LL_TIM_DMABURST_LENGTH_8TRANSFERS – LL_TIM_DMABURST_LENGTH_9TRANSFERS – LL_TIM_DMABURST_LENGTH_10TRANSFERS – LL_TIM_DMABURST_LENGTH_11TRANSFERS – LL_TIM_DMABURST_LENGTH_12TRANSFERS – LL_TIM_DMABURST_LENGTH_13TRANSFERS – LL_TIM_DMABURST_LENGTH_14TRANSFERS – LL_TIM_DMABURST_LENGTH_15TRANSFERS – LL_TIM_DMABURST_LENGTH_16TRANSFERS – LL_TIM_DMABURST_LENGTH_17TRANSFERS – LL_TIM_DMABURST_LENGTH_18TRANSFERS
Return values	• **None:**
Notes	• Macro IS_TIM_DMABURST_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the DMA burst mode.
Reference Manual to LL API cross reference:	• DCR DBL LL_TIM_ConfigDMABurst • DCR DBA LL_TIM_ConfigDMABurst

LL_TIM_SetRemap

Function name	**__STATIC_INLINE void LL_TIM_SetRemap (TIM_TypeDef * TIMx, uint32_t Remap)**
Function description	Remap TIM inputs (input channel, internal/external triggers).
Parameters	• **TIMx:** Timer instance • **Remap:** Remap param depends on the TIMx. Description available only in CHM version of the User Manual (not in .pdf). Otherwise see Reference Manual description of OR registers.
Return values	• **None:**
Notes	• Macro IS_TIM_REMAP_INSTANCE(TIMx) can be used to check whether or not a some timer inputs can be remapped.

LL_TIM_SetOCRefClearInputSource

Function name	**__STATIC_INLINE void LL_TIM_SetOCRefClearInputSource (TIM_TypeDef * TIMx, uint32_t OCRefClearInputSource)**
Function description	Set the OCREF clear input source.
Parameters	• **TIMx:** Timer instance • **OCRefClearInputSource:** This parameter can be one of the following values: – LL_TIM_OCREF_CLR_INT_NC – LL_TIM_OCREF_CLR_INT_ETR
Return values	• **None:**
Notes	• The OCxREF signal of a given channel can be cleared when a high level is applied on the OCREF_CLR_INPUT • This function can only be used in Output compare and PWM

	modes.
Reference Manual to LL API cross reference:	• SMCR OCCS LL_TIM_SetOCRefClearInputSource

LL_TIM_ClearFlag_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_UPDATE (TIM_TypeDef * TIMx)**
Function description	Clear the update interrupt flag (UIF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR UIF LL_TIM_ClearFlag_UPDATE

LL_TIM_IsActiveFlag_UPDATE

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_UPDATE (TIM_TypeDef * TIMx)**
Function description	Indicate whether update interrupt flag (UIF) is set (update interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR UIF LL_TIM_IsActiveFlag_UPDATE

LL_TIM_ClearFlag_CC1

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC1 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 1 interrupt flag (CC1F).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC1IF LL_TIM_ClearFlag_CC1

LL_TIM_IsActiveFlag_CC1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 1 interrupt flag (CC1F) is set (Capture/Compare 1 interrupt is pending).
Parameters	• **TIMx:** Timer instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC1IF LL_TIM_IsActiveFlag_CC1

LL_TIM_ClearFlag_CC2

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC2 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 2 interrupt flag (CC2F).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC2IF LL_TIM_ClearFlag_CC2

LL_TIM_IsActiveFlag_CC2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 2 interrupt flag (CC2F) is set (Capture/Compare 2 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC2IF LL_TIM_IsActiveFlag_CC2

LL_TIM_ClearFlag_CC3

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC3 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 3 interrupt flag (CC3F).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC3IF LL_TIM_ClearFlag_CC3

LL_TIM_IsActiveFlag_CC3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 3 interrupt flag (CC3F) is set (Capture/Compare 3 interrupt is pending).
Parameters	• **TIMx:** Timer instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC3IF LL_TIM_IsActiveFlag_CC3

LL_TIM_ClearFlag_CC4

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC4 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 4 interrupt flag (CC4F).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC4IF LL_TIM_ClearFlag_CC4

LL_TIM_IsActiveFlag_CC4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 4 interrupt flag (CC4F) is set (Capture/Compare 4 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC4IF LL_TIM_IsActiveFlag_CC4

LL_TIM_ClearFlag_CC5

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC5 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 5 interrupt flag (CC5F).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC5IF LL_TIM_ClearFlag_CC5

LL_TIM_IsActiveFlag_CC5

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC5 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 5 interrupt flag (CC5F) is set (Capture/Compare 5 interrupt is pending).
Parameters	• **TIMx:** Timer instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC5IF LL_TIM_IsActiveFlag_CC5

LL_TIM_ClearFlag_CC6

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC6 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 6 interrupt flag (CC6F).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC6IF LL_TIM_ClearFlag_CC6

LL_TIM_IsActiveFlag_CC6

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC6 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 6 interrupt flag (CC6F) is set (Capture/Compare 6 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC6IF LL_TIM_IsActiveFlag_CC6

LL_TIM_ClearFlag_COM

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_COM (TIM_TypeDef * TIMx)**
Function description	Clear the commutation interrupt flag (COMIF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR COMIF LL_TIM_ClearFlag_COM

LL_TIM_IsActiveFlag_COM

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_COM (TIM_TypeDef * TIMx)**
Function description	Indicate whether commutation interrupt flag (COMIF) is set (commutation interrupt is pending).
Parameters	• **TIMx:** Timer instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR COMIF LL_TIM_IsActiveFlag_COM

LL_TIM_ClearFlag_TRIG

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_TRIG (TIM_TypeDef * TIMx)**
Function description	Clear the trigger interrupt flag (TIF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR TIF LL_TIM_ClearFlag_TRIG

LL_TIM_IsActiveFlag_TRIG

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TRIG (TIM_TypeDef * TIMx)**
Function description	Indicate whether trigger interrupt flag (TIF) is set (trigger interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR TIF LL_TIM_IsActiveFlag_TRIG

LL_TIM_ClearFlag_BRK

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_BRK (TIM_TypeDef * TIMx)**
Function description	Clear the break interrupt flag (BIF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR BIF LL_TIM_ClearFlag_BRK

LL_TIM_IsActiveFlag_BRK

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK (TIM_TypeDef * TIMx)**
Function description	Indicate whether break interrupt flag (BIF) is set (break interrupt is pending).
Parameters	• **TIMx:** Timer instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR BIF LL_TIM_IsActiveFlag_BRK

LL_TIM_ClearFlag_BRK2

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_BRK2 (TIM_TypeDef * TIMx)**
Function description	Clear the break 2 interrupt flag (B2IF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR B2IF LL_TIM_ClearFlag_BRK2

LL_TIM_IsActiveFlag_BRK2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK2 (TIM_TypeDef * TIMx)**
Function description	Indicate whether break 2 interrupt flag (B2IF) is set (break 2 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR B2IF LL_TIM_IsActiveFlag_BRK2

LL_TIM_ClearFlag_CC1OVR

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC1OVR (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 1 over-capture interrupt flag (CC1OF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC1OF LL_TIM_ClearFlag_CC1OVR

LL_TIM_IsActiveFlag_CC1OVR

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1OVR (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 1 over-capture interrupt flag (CC1OF) is set (Capture/Compare 1 interrupt is pending).
Parameters	• **TIMx:** Timer instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC1OF LL_TIM_IsActiveFlag_CC1OVR

LL_TIM_ClearFlag_CC2OVR

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC2OVR (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 2 over-capture interrupt flag (CC2OF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC2OF LL_TIM_ClearFlag_CC2OVR

LL_TIM_IsActiveFlag_CC2OVR

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2OVR (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 2 over-capture interrupt flag (CC2OF) is set (Capture/Compare 2 over-capture interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC2OF LL_TIM_IsActiveFlag_CC2OVR

LL_TIM_ClearFlag_CC3OVR

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC3OVR (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 3 over-capture interrupt flag (CC3OF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC3OF LL_TIM_ClearFlag_CC3OVR

LL_TIM_IsActiveFlag_CC3OVR

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3OVR (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 3 over-capture interrupt flag (CC3OF) is set (Capture/Compare 3 over-capture interrupt is pending).

Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC3OF LL_TIM_IsActiveFlag_CC3OVR

LL_TIM_ClearFlag_CC4OVR

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC4OVR (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 4 over-capture interrupt flag (CC4OF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC4OF LL_TIM_ClearFlag_CC4OVR

LL_TIM_IsActiveFlag_CC4OVR

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4OVR (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 4 over-capture interrupt flag (CC4OF) is set (Capture/Compare 4 over-capture interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC4OF LL_TIM_IsActiveFlag_CC4OVR

LL_TIM_ClearFlag_SYSBRK

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_SYSBRK (TIM_TypeDef * TIMx)**
Function description	Clear the system break interrupt flag (SBIF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR SBIF LL_TIM_ClearFlag_SYSBRK

LL_TIM_IsActiveFlag_SYSBRK

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_SYSBRK (TIM_TypeDef * TIMx)**
Function description	Indicate whether system break interrupt flag (SBIF) is set (system

	break interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR SBIF LL_TIM_IsActiveFlag_SYSBRK

LL_TIM_EnableIT_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_UPDATE (TIM_TypeDef * TIMx)**
Function description	Enable update interrupt (UIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER UIE LL_TIM_EnableIT_UPDATE

LL_TIM_DisableIT_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_UPDATE (TIM_TypeDef * TIMx)**
Function description	Disable update interrupt (UIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER UIE LL_TIM_DisableIT_UPDATE

LL_TIM_IsEnabledIT_UPDATE

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_UPDATE (TIM_TypeDef * TIMx)**
Function description	Indicates whether the update interrupt (UIE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER UIE LL_TIM_IsEnabledIT_UPDATE

LL_TIM_EnableIT_CC1

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC1 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 1 interrupt (CC1IE).

Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC1IE LL_TIM_EnableIT_CC1

LL_TIM_DisableIT_CC1

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC1 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 1 interrupt (CC1IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC1IE LL_TIM_DisableIT_CC1

LL_TIM_IsEnabledIT_CC1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC1 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC1IE LL_TIM_IsEnabledIT_CC1

LL_TIM_EnableIT_CC2

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC2 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 2 interrupt (CC2IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC2IE LL_TIM_EnableIT_CC2

LL_TIM_DisableIT_CC2

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC2 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 2 interrupt (CC2IE).

Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC2IE LL_TIM_DisableIT_CC2

LL_TIM_IsEnabledIT_CC2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC2 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC2IE LL_TIM_IsEnabledIT_CC2

LL_TIM_EnableIT_CC3

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC3 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 3 interrupt (CC3IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC3IE LL_TIM_EnableIT_CC3

LL_TIM_DisableIT_CC3

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC3 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 3 interrupt (CC3IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC3IE LL_TIM_DisableIT_CC3

LL_TIM_IsEnabledIT_CC3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC3 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 3 interrupt (CC3IE) is enabled.

Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC3IE LL_TIM_IsEnabledIT_CC3

LL_TIM_EnableIT_CC4

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC4 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 4 interrupt (CC4IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC4IE LL_TIM_EnableIT_CC4

LL_TIM_DisableIT_CC4

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC4 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 4 interrupt (CC4IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC4IE LL_TIM_DisableIT_CC4

LL_TIM_IsEnabledIT_CC4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC4 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 4 interrupt (CC4IE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC4IE LL_TIM_IsEnabledIT_CC4

LL_TIM_EnableIT_COM

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_COM (TIM_TypeDef * TIMx)**
Function description	Enable commutation interrupt (COMIE).

Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER COMIE LL_TIM_EnableIT_COM

LL_TIM_DisableIT_COM

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_COM (TIM_TypeDef * TIMx)**
Function description	Disable commutation interrupt (COMIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER COMIE LL_TIM_DisableIT_COM

LL_TIM_IsEnabledIT_COM

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_COM (TIM_TypeDef * TIMx)**
Function description	Indicates whether the commutation interrupt (COMIE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER COMIE LL_TIM_IsEnabledIT_COM

LL_TIM_EnableIT_TRIG

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_TRIG (TIM_TypeDef * TIMx)**
Function description	Enable trigger interrupt (TIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER TIE LL_TIM_EnableIT_TRIG

LL_TIM_DisableIT_TRIG

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_TRIG (TIM_TypeDef * TIMx)**
Function description	Disable trigger interrupt (TIE).
Parameters	• **TIMx:** Timer instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER TIE LL_TIM_DisableIT_TRIG

LL_TIM_IsEnabledIT_TRIG

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TRIG (TIM_TypeDef * TIMx)**
Function description	Indicates whether the trigger interrupt (TIE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER TIE LL_TIM_IsEnabledIT_TRIG

LL_TIM_EnableIT_BRK

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_BRK (TIM_TypeDef * TIMx)**
Function description	Enable break interrupt (BIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER BIE LL_TIM_EnableIT_BRK

LL_TIM_DisableIT_BRK

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_BRK (TIM_TypeDef * TIMx)**
Function description	Disable break interrupt (BIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER BIE LL_TIM_DisableIT_BRK

LL_TIM_IsEnabledIT_BRK

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_BRK (TIM_TypeDef * TIMx)**
Function description	Indicates whether the break interrupt (BIE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• DIER BIE LL_TIM_IsEnabledIT_BRK

LL_TIM_EnableDMAReq_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_UPDATE (TIM_TypeDef * TIMx)**
Function description	Enable update DMA request (UDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER UDE LL_TIM_EnableDMAReq_UPDATE

LL_TIM_DisableDMAReq_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_UPDATE (TIM_TypeDef * TIMx)**
Function description	Disable update DMA request (UDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER UDE LL_TIM_DisableDMAReq_UPDATE

LL_TIM_IsEnabledDMAReq_UPDATE

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_UPDATE (TIM_TypeDef * TIMx)**
Function description	Indicates whether the update DMA request (UDE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER UDE LL_TIM_IsEnabledDMAReq_UPDATE

LL_TIM_EnableDMAReq_CC1

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC1 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 1 DMA request (CC1DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross	• DIER CC1DE LL_TIM_EnableDMAReq_CC1

reference:

LL_TIM_DisableDMAReq_CC1

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC1 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 1 DMA request (CC1DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC1DE LL_TIM_DisableDMAReq_CC1

LL_TIM_IsEnabledDMAReq_CC1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC1 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC1DE LL_TIM_IsEnabledDMAReq_CC1

LL_TIM_EnableDMAReq_CC2

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC2 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 2 DMA request (CC2DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC2DE LL_TIM_EnableDMAReq_CC2

LL_TIM_DisableDMAReq_CC2

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC2 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 2 DMA request (CC2DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross	• DIER CC2DE LL_TIM_DisableDMAReq_CC2

reference:

LL_TIM_IsEnabledDMAReq_CC2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC2 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC2DE LL_TIM_IsEnabledDMAReq_CC2

LL_TIM_EnableDMAReq_CC3

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC3 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 3 DMA request (CC3DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC3DE LL_TIM_EnableDMAReq_CC3

LL_TIM_DisableDMAReq_CC3

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC3 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 3 DMA request (CC3DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC3DE LL_TIM_DisableDMAReq_CC3

LL_TIM_IsEnabledDMAReq_CC3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC3 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 3 DMA request (CC3DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• DIER CC3DE LL_TIM_IsEnabledDMAReq_CC3

reference:

LL_TIM_EnableDMAReq_CC4

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC4 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 4 DMA request (CC4DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC4DE LL_TIM_EnableDMAReq_CC4

LL_TIM_DisableDMAReq_CC4

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC4 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 4 DMA request (CC4DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC4DE LL_TIM_DisableDMAReq_CC4

LL_TIM_IsEnabledDMAReq_CC4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC4 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 4 DMA request (CC4DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC4DE LL_TIM_IsEnabledDMAReq_CC4

LL_TIM_EnableDMAReq_COM

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_COM (TIM_TypeDef * TIMx)**
Function description	Enable commutation DMA request (COMDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross	• DIER COMDE LL_TIM_EnableDMAReq_COM

reference:

LL_TIM_DisableDMAReq_COM

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_COM (TIM_TypeDef * TIMx)**
Function description	Disable commutation DMA request (COMDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER COMDE LL_TIM_DisableDMAReq_COM

LL_TIM_IsEnabledDMAReq_COM

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_COM (TIM_TypeDef * TIMx)**
Function description	Indicates whether the commutation DMA request (COMDE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER COMDE LL_TIM_IsEnabledDMAReq_COM

LL_TIM_EnableDMAReq_TRIG

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)**
Function description	Enable trigger interrupt (TDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER TDE LL_TIM_EnableDMAReq_TRIG

LL_TIM_DisableDMAReq_TRIG

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_TRIG (TIM_TypeDef * TIMx)**
Function description	Disable trigger interrupt (TDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross	• DIER TDE LL_TIM_DisableDMAReq_TRIG

reference:

LL_TIM_IsEnabledDMAReq_TRIG

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_TRIG (TIM_TypeDef * TIMx)**
Function description	Indicates whether the trigger interrupt (TDE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER TDE LL_TIM_IsEnabledDMAReq_TRIG

LL_TIM_GenerateEvent_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_UPDATE (TIM_TypeDef * TIMx)**
Function description	Generate an update event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR UG LL_TIM_GenerateEvent_UPDATE

LL_TIM_GenerateEvent_CC1

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC1 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 1 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR CC1G LL_TIM_GenerateEvent_CC1

LL_TIM_GenerateEvent_CC2

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 2 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR CC2G LL_TIM_GenerateEvent_CC2

LL_TIM_GenerateEvent_CC3

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 3 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR CC3G LL_TIM_GenerateEvent_CC3

LL_TIM_GenerateEvent_CC4

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 4 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR CC4G LL_TIM_GenerateEvent_CC4

LL_TIM_GenerateEvent_COM

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_COM (TIM_TypeDef * TIMx)**
Function description	Generate commutation event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR COMG LL_TIM_GenerateEvent_COM

LL_TIM_GenerateEvent_TRIG

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)**
Function description	Generate trigger event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR TG LL_TIM_GenerateEvent_TRIG

LL_TIM_GenerateEvent_BRK

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_BRK (TIM_TypeDef * TIMx)**
Function description	Generate break event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR BG LL_TIM_GenerateEvent_BRK

LL_TIM_GenerateEvent_BRK2

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_BRK2 (TIM_TypeDef * TIMx)**
Function description	Generate break 2 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR B2G LL_TIM_GenerateEvent_BRK2

LL_TIM_DeInit

Function name	**ErrorStatus LL_TIM_DeInit (TIM_TypeDef * TIMx)**
Function description	Set TIMx registers to their reset values.
Parameters	• **TIMx:** Timer instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx registers are de-initialized – ERROR: invalid TIMx instance

LL_TIM_StructInit

Function name	**void LL_TIM_StructInit (LL_TIM_InitTypeDef * TIM_InitStruct)**
Function description	Set the fields of the time base unit configuration data structure to their default values.
Parameters	• **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (time base unit configuration data structure)
Return values	• **None:**

LL_TIM_Init

Function name	**ErrorStatus LL_TIM_Init (TIM_TypeDef * TIMx, LL_TIM_InitTypeDef * TIM_InitStruct)**
Function description	Configure the TIMx time base unit.
Parameters	• **TIMx:** Timer Instance

	• **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (TIMx time base unit configuration data structure)
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx registers are de-initialized – ERROR: not applicable

LL_TIM_OC_StructInit

Function name	**void LL_TIM_OC_StructInit (LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)**
Function description	Set the fields of the TIMx output channel configuration data structure to their default values.
Parameters	• **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (the output channel configuration data structure)
Return values	• **None:**

LL_TIM_OC_Init

Function name	**ErrorStatus LL_TIM_OC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)**
Function description	Configure the TIMx output channel.
Parameters	• **TIMx:** Timer Instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 – LL_TIM_CHANNEL_CH5 – LL_TIM_CHANNEL_CH6 • **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (TIMx output channel configuration data structure)
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx output channel is initialized – ERROR: TIMx output channel is not initialized

LL_TIM_IC_StructInit

Function name	**void LL_TIM_IC_StructInit (LL_TIM_IC_InitTypeDef * TIM_ICInitStruct)**
Function description	Set the fields of the TIMx input channel configuration data structure to their default values.
Parameters	• **TIM_ICInitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (the input channel configuration data structure)
Return values	• **None:**

LL_TIM_IC_Init

Function name	**ErrorStatus LL_TIM_IC_Init (TIM_TypeDef * TIMx, uint32_t**

	Channel, LL_TIM_IC_InitTypeDef * TIM_IC_InitStruct)
Function description	Configure the TIMx input channel.
Parameters	• **TIMx:** Timer Instance • **Channel:** This parameter can be one of the following values: – LL_TIM_CHANNEL_CH1 – LL_TIM_CHANNEL_CH2 – LL_TIM_CHANNEL_CH3 – LL_TIM_CHANNEL_CH4 • **TIM_IC_InitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (TIMx input channel configuration data structure)
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx output channel is initialized – ERROR: TIMx output channel is not initialized

LL_TIM_ENCODER_StructInit

Function name	**void LL_TIM_ENCODER_StructInit (LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)**
Function description	Fills each TIM_EncoderInitStruct field with its default value.
Parameters	• **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (encoder interface configuration data structure)
Return values	• **None:**

LL_TIM_ENCODER_Init

Function name	**ErrorStatus LL_TIM_ENCODER_Init (TIM_TypeDef * TIMx, LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)**
Function description	Configure the encoder interface of the timer instance.
Parameters	• **TIMx:** Timer Instance • **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (TIMx encoder interface configuration data structure)
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx registers are de-initialized – ERROR: not applicable

LL_TIM_HALLSENSOR_StructInit

Function name	**void LL_TIM_HALLSENSOR_StructInit (LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)**
Function description	Set the fields of the TIMx Hall sensor interface configuration data structure to their default values.
Parameters	• **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (HALL sensor interface configuration data structure)

Return values	• **None:**

LL_TIM_HALLSENSOR_Init

Function name	**ErrorStatus LL_TIM_HALLSENSOR_Init (TIM_TypeDef * TIMx, LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)**
Function description	Configure the Hall sensor interface of the timer instance.
Parameters	• **TIMx:** Timer Instance • **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (TIMx HALL sensor interface configuration data structure)
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: TIMx registers are de-initialized – ERROR: not applicable
Notes	• TIMx CH1, CH2 and CH3 inputs connected through a XOR to the TI1 input channel • TIMx slave mode controller is configured in reset mode. Selected internal trigger is TI1F_ED. • Channel 1 is configured as input, IC1 is mapped on TRC. • Captured value stored in TIMx_CCR1 correspond to the time elapsed between 2 changes on the inputs. It gives information about motor speed. • Channel 2 is configured in output PWM 2 mode. • Compare value stored in TIMx_CCR2 corresponds to the commutation delay. • OC2REF is selected as trigger output on TRGO. • LL_TIM_IC_POLARITY_BOTHEDGE must not be used for TI1 when it is used when TIMx operates in Hall sensor interface mode.

LL_TIM_BDTR_StructInit

Function name	**void LL_TIM_BDTR_StructInit (LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)**
Function description	Set the fields of the Break and Dead Time configuration data structure to their default values.
Parameters	• **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)
Return values	• **None:**

LL_TIM_BDTR_Init

Function name	**ErrorStatus LL_TIM_BDTR_Init (TIM_TypeDef * TIMx, LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)**
Function description	Configure the Break and Dead Time feature of the timer instance.
Parameters	• **TIMx:** Timer Instance • **TIM_BDTRInitStruct:** pointer to a

		LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)
Return values	•	**An:** ErrorStatus enumeration value: – SUCCESS: Break and Dead Time is initialized – ERROR: not applicable
Notes	•	As the bits BK2P, BK2E, BK2F[3:0], BKF[3:0], AOE, BKP, BKE, OSSI, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BDTR register.
	•	Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
	•	Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

96.3 TIM Firmware driver defines

96.3.1 TIM

Active Input Selection

LL_TIM_ACTIVEINPUT_DIRECTTI	ICx is mapped on TIx
LL_TIM_ACTIVEINPUT_INDIRECTTI	ICx is mapped on TIy
LL_TIM_ACTIVEINPUT_TRC	ICx is mapped on TRC

Automatic output enable

LL_TIM_AUTOMATICOUTPUT_DISABLE	MOE can be set only by software
LL_TIM_AUTOMATICOUTPUT_ENABLE	MOE can be set by software or automatically at the next update event

BKIN POLARITY

LL_TIM_BKIN_POLARITY_LOW	BRK BKIN input is active low
LL_TIM_BKIN_POLARITY_HIGH	BRK BKIN input is active high

BKIN SOURCE

LL_TIM_BKIN_SOURCE_BKIN	BKIN input from AF controller
LL_TIM_BKIN_SOURCE_BKCOMP1	internal signal: COMP1 output
LL_TIM_BKIN_SOURCE_BKCOMP2	internal signal: COMP2 output
LL_TIM_BKIN_SOURCE_DF1BK	internal signal: DFSDM1 break output

Break2 Enable

LL_TIM_BREAK2_DISABLE	Break2 function disabled
LL_TIM_BREAK2_ENABLE	Break2 function enabled

BREAK2 FILTER

LL_TIM_BREAK2_FILTER_FDIV1	No filter, BRK acts asynchronously
LL_TIM_BREAK2_FILTER_FDIV1_N2	fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK2_FILTER_FDIV1_N4	fSAMPLING=fCK_INT, N=4
LL_TIM_BREAK2_FILTER_FDIV1_N8	fSAMPLING=fCK_INT, N=8
LL_TIM_BREAK2_FILTER_FDIV2_N6	fSAMPLING=fDTS/2, N=6
LL_TIM_BREAK2_FILTER_FDIV2_N8	fSAMPLING=fDTS/2, N=8
LL_TIM_BREAK2_FILTER_FDIV4_N6	fSAMPLING=fDTS/4, N=6
LL_TIM_BREAK2_FILTER_FDIV4_N8	fSAMPLING=fDTS/4, N=8
LL_TIM_BREAK2_FILTER_FDIV8_N6	fSAMPLING=fDTS/8, N=6
LL_TIM_BREAK2_FILTER_FDIV8_N8	fSAMPLING=fDTS/8, N=8
LL_TIM_BREAK2_FILTER_FDIV16_N5	fSAMPLING=fDTS/16, N=5
LL_TIM_BREAK2_FILTER_FDIV16_N6	fSAMPLING=fDTS/16, N=6
LL_TIM_BREAK2_FILTER_FDIV16_N8	fSAMPLING=fDTS/16, N=8
LL_TIM_BREAK2_FILTER_FDIV32_N5	fSAMPLING=fDTS/32, N=5
LL_TIM_BREAK2_FILTER_FDIV32_N6	fSAMPLING=fDTS/32, N=6
LL_TIM_BREAK2_FILTER_FDIV32_N8	fSAMPLING=fDTS/32, N=8

BREAK2 POLARITY

LL_TIM_BREAK2_POLARITY_LOW	Break input BRK2 is active low
LL_TIM_BREAK2_POLARITY_HIGH	Break input BRK2 is active high

Break Enable

LL_TIM_BREAK_DISABLE	Break function disabled
LL_TIM_BREAK_ENABLE	Break function enabled

break filter

LL_TIM_BREAK_FILTER_FDIV1	No filter, BRK acts asynchronously
LL_TIM_BREAK_FILTER_FDIV1_N2	fSAMPLING=fCK_INT, N=2
LL_TIM_BREAK_FILTER_FDIV1_N4	fSAMPLING=fCK_INT, N=4
LL_TIM_BREAK_FILTER_FDIV1_N8	fSAMPLING=fCK_INT, N=8
LL_TIM_BREAK_FILTER_FDIV2_N6	fSAMPLING=fDTS/2, N=6
LL_TIM_BREAK_FILTER_FDIV2_N8	fSAMPLING=fDTS/2, N=8
LL_TIM_BREAK_FILTER_FDIV4_N6	fSAMPLING=fDTS/4, N=6
LL_TIM_BREAK_FILTER_FDIV4_N8	fSAMPLING=fDTS/4, N=8
LL_TIM_BREAK_FILTER_FDIV8_N6	fSAMPLING=fDTS/8, N=6
LL_TIM_BREAK_FILTER_FDIV8_N8	fSAMPLING=fDTS/8, N=8
LL_TIM_BREAK_FILTER_FDIV16_N5	fSAMPLING=fDTS/16, N=5
LL_TIM_BREAK_FILTER_FDIV16_N6	fSAMPLING=fDTS/16, N=6
LL_TIM_BREAK_FILTER_FDIV16_N8	fSAMPLING=fDTS/16, N=8
LL_TIM_BREAK_FILTER_FDIV32_N5	fSAMPLING=fDTS/32, N=5
LL_TIM_BREAK_FILTER_FDIV32_N6	fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK_FILTER_FDIV32_N8	fSAMPLING=fDTS/32, N=8

BREAK INPUT

LL_TIM_BREAK_INPUT_BKIN	TIMx_BKIN input
LL_TIM_BREAK_INPUT_BKIN2	TIMx_BKIN2 input

break polarity

LL_TIM_BREAK_POLARITY_LOW	Break input BRK is active low
LL_TIM_BREAK_POLARITY_HIGH	Break input BRK is active high

Capture Compare DMA Request

LL_TIM_CCDMAREQUEST_CC	CCx DMA request sent when CCx event occurs
LL_TIM_CCDMAREQUEST_UPDATE	CCx DMA requests sent when update event occurs

Capture Compare Update Source

LL_TIM_CCUPDATESOURCE_COMG_ONLY	Capture/compare control bits are updated by setting the COMG bit only
LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI	Capture/compare control bits are updated by setting the COMG bit or when a rising edge occurs on trigger input (TRGI)

Channel

LL_TIM_CHANNEL_CH1	Timer input/output channel 1
LL_TIM_CHANNEL_CH1N	Timer complementary output channel 1
LL_TIM_CHANNEL_CH2	Timer input/output channel 2
LL_TIM_CHANNEL_CH2N	Timer complementary output channel 2
LL_TIM_CHANNEL_CH3	Timer input/output channel 3
LL_TIM_CHANNEL_CH3N	Timer complementary output channel 3
LL_TIM_CHANNEL_CH4	Timer input/output channel 4
LL_TIM_CHANNEL_CH5	Timer output channel 5
LL_TIM_CHANNEL_CH6	Timer output channel 6

Clock Division

LL_TIM_CLOCKDIVISION_DIV1	tDTS=tCK_INT
LL_TIM_CLOCKDIVISION_DIV2	tDTS=2*tCK_INT
LL_TIM_CLOCKDIVISION_DIV4	tDTS=4*tCK_INT

Clock Source

LL_TIM_CLOCKSOURCE_INTERNAL	The timer is clocked by the internal clock provided from the RCC
LL_TIM_CLOCKSOURCE_EXT_MODE1	Counter counts at each rising or falling edge on a selected inpu t
LL_TIM_CLOCKSOURCE_EXT_MODE2	Counter counts at each rising or falling edge on

	the external trigger input ETR

Counter Direction

LL_TIM_COUNTERDIRECTION_UP	Timer counter counts up
LL_TIM_COUNTERDIRECTION_DOWN	Timer counter counts down

Counter Mode

LL_TIM_COUNTERMODE_UP	Counter used as upcounter
LL_TIM_COUNTERMODE_DOWN	Counter used as downcounter
LL_TIM_COUNTERMODE_CENTER_UP	The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting down.
LL_TIM_COUNTERMODE_CENTER_DOWN	The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up
LL_TIM_COUNTERMODE_CENTER_UP_DOWN	The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up or down.

DMA Burst Base Address

LL_TIM_DMABURST_BASEADDR_CR1	TIMx_CR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CR2	TIMx_CR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_SMCR	TIMx_SMCR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_DIER	TIMx_DIER register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_SR	TIMx_SR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_EGR	TIMx_EGR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCMR1	TIMx_CCMR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCMR2	TIMx_CCMR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCER	TIMx_CCER register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CNT	TIMx_CNT register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_PSC	TIMx_PSC register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_ARR	TIMx_ARR register is the DMA base

	address for DMA burst
LL_TIM_DMABURST_BASEADDR_RCR	TIMx_RCR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR1	TIMx_CCR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR2	TIMx_CCR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR3	TIMx_CCR3 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR4	TIMx_CCR4 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_BDTR	TIMx_BDTR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCMR3	TIMx_CCMR3 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR5	TIMx_CCR5 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR6	TIMx_CCR6 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_OR1	TIMx_OR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_OR2	TIMx_OR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_OR3	TIMx_OR3 register is the DMA base address for DMA burst

DMA Burst Length

LL_TIM_DMABURST_LENGTH_1TRANSFER	Transfer is done to 1 register starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_2TRANSFERS	Transfer is done to 2 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_3TRANSFERS	Transfer is done to 3 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_4TRANSFERS	Transfer is done to 4 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_5TRANSFERS	Transfer is done to 5 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_6TRANSFERS	Transfer is done to 6 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_7TRANSFERS	Transfer is done to 7 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_8TRANSFERS	Transfer is done to 1 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_9TRANSFERS	Transfer is done to 9 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_10TRANSFERS	Transfer is done to 10 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_11TRANSFERS	Transfer is done to 11 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_12TRANSFERS	Transfer is done to 12 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_13TRANSFERS	Transfer is done to 13 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_14TRANSFERS	Transfer is done to 14 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_15TRANSFERS	Transfer is done to 15 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_16TRANSFERS	Transfer is done to 16 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_17TRANSFERS	Transfer is done to 17 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_18TRANSFERS	Transfer is done to 18 registers starting from the DMA burst base address

Encoder Mode

LL_TIM_ENCODERMODE_X2_TI1	Encoder mode 1 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level
LL_TIM_ENCODERMODE_X2_TI2	Encoder mode 2 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level
LL_TIM_ENCODERMODE_X4_TI12	Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input l

External Trigger Source

LL_TIM_ETRSOURCE_LEGACY	ETR legacy mode
LL_TIM_ETRSOURCE_COMP1	COMP1 output connected to ETR input
LL_TIM_ETRSOURCE_COMP2	COMP2 output connected to ETR input

External Trigger Filter

LL_TIM_ETR_FILTER_FDIV1	No filter, sampling is done at fDTS
LL_TIM_ETR_FILTER_FDIV1_N2	fSAMPLING=fCK_INT, N=2
LL_TIM_ETR_FILTER_FDIV1_N4	fSAMPLING=fCK_INT, N=4
LL_TIM_ETR_FILTER_FDIV1_N8	fSAMPLING=fCK_INT, N=8
LL_TIM_ETR_FILTER_FDIV2_N6	fSAMPLING=fDTS/2, N=6
LL_TIM_ETR_FILTER_FDIV2_N8	fSAMPLING=fDTS/2, N=8
LL_TIM_ETR_FILTER_FDIV4_N6	fSAMPLING=fDTS/4, N=6
LL_TIM_ETR_FILTER_FDIV4_N8	fSAMPLING=fDTS/4, N=8
LL_TIM_ETR_FILTER_FDIV8_N6	fSAMPLING=fDTS/8, N=8
LL_TIM_ETR_FILTER_FDIV8_N8	fSAMPLING=fDTS/16, N=5

LL_TIM_ETR_FILTER_FDIV16_N5	fSAMPLING=fDTS/16, N=6
LL_TIM_ETR_FILTER_FDIV16_N6	fSAMPLING=fDTS/16, N=8
LL_TIM_ETR_FILTER_FDIV16_N8	fSAMPLING=fDTS/16, N=5
LL_TIM_ETR_FILTER_FDIV32_N5	fSAMPLING=fDTS/32, N=5
LL_TIM_ETR_FILTER_FDIV32_N6	fSAMPLING=fDTS/32, N=6
LL_TIM_ETR_FILTER_FDIV32_N8	fSAMPLING=fDTS/32, N=8

External Trigger Polarity

LL_TIM_ETR_POLARITY_NONINVERTED	ETR is non-inverted, active at high level or rising edge
LL_TIM_ETR_POLARITY_INVERTED	ETR is inverted, active at low level or falling edge

External Trigger Prescaler

LL_TIM_ETR_PRESCALER_DIV1	ETR prescaler OFF
LL_TIM_ETR_PRESCALER_DIV2	ETR frequency is divided by 2
LL_TIM_ETR_PRESCALER_DIV4	ETR frequency is divided by 4
LL_TIM_ETR_PRESCALER_DIV8	ETR frequency is divided by 8

Get Flags Defines

LL_TIM_SR_UIF	Update interrupt flag
LL_TIM_SR_CC1IF	Capture/compare 1 interrupt flag
LL_TIM_SR_CC2IF	Capture/compare 2 interrupt flag
LL_TIM_SR_CC3IF	Capture/compare 3 interrupt flag
LL_TIM_SR_CC4IF	Capture/compare 4 interrupt flag
LL_TIM_SR_CC5IF	Capture/compare 5 interrupt flag
LL_TIM_SR_CC6IF	Capture/compare 6 interrupt flag
LL_TIM_SR_COMIF	COM interrupt flag
LL_TIM_SR_TIF	Trigger interrupt flag
LL_TIM_SR_BIF	Break interrupt flag
LL_TIM_SR_B2IF	Second break interrupt flag
LL_TIM_SR_CC1OF	Capture/Compare 1 overcapture flag
LL_TIM_SR_CC2OF	Capture/Compare 2 overcapture flag
LL_TIM_SR_CC3OF	Capture/Compare 3 overcapture flag
LL_TIM_SR_CC4OF	Capture/Compare 4 overcapture flag
LL_TIM_SR_SBIF	System Break interrupt flag

GROUPCH5

LL_TIM_GROUPCH5_NONE	No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC
LL_TIM_GROUPCH5_OC1REFC	OC1REFC is the logical AND of OC1REFC and OC5REF

LL_TIM_GROUPCH5_OC2REFC	OC2REFC is the logical AND of OC2REFC and OC5REF
LL_TIM_GROUPCH5_OC3REFC	OC3REFC is the logical AND of OC3REFC and OC5REF

Input Configuration Prescaler

LL_TIM_ICPSC_DIV1	No prescaler, capture is done each time an edge is detected on the capture input
LL_TIM_ICPSC_DIV2	Capture is done once every 2 events
LL_TIM_ICPSC_DIV4	Capture is done once every 4 events
LL_TIM_ICPSC_DIV8	Capture is done once every 8 events

Input Configuration Filter

LL_TIM_IC_FILTER_FDIV1	No filter, sampling is done at fDTS
LL_TIM_IC_FILTER_FDIV1_N2	fSAMPLING=fCK_INT, N=2
LL_TIM_IC_FILTER_FDIV1_N4	fSAMPLING=fCK_INT, N=4
LL_TIM_IC_FILTER_FDIV1_N8	fSAMPLING=fCK_INT, N=8
LL_TIM_IC_FILTER_FDIV2_N6	fSAMPLING=fDTS/2, N=6
LL_TIM_IC_FILTER_FDIV2_N8	fSAMPLING=fDTS/2, N=8
LL_TIM_IC_FILTER_FDIV4_N6	fSAMPLING=fDTS/4, N=6
LL_TIM_IC_FILTER_FDIV4_N8	fSAMPLING=fDTS/4, N=8
LL_TIM_IC_FILTER_FDIV8_N6	fSAMPLING=fDTS/8, N=6
LL_TIM_IC_FILTER_FDIV8_N8	fSAMPLING=fDTS/8, N=8
LL_TIM_IC_FILTER_FDIV16_N5	fSAMPLING=fDTS/16, N=5
LL_TIM_IC_FILTER_FDIV16_N6	fSAMPLING=fDTS/16, N=6
LL_TIM_IC_FILTER_FDIV16_N8	fSAMPLING=fDTS/16, N=8
LL_TIM_IC_FILTER_FDIV32_N5	fSAMPLING=fDTS/32, N=5
LL_TIM_IC_FILTER_FDIV32_N6	fSAMPLING=fDTS/32, N=6
LL_TIM_IC_FILTER_FDIV32_N8	fSAMPLING=fDTS/32, N=8

Input Configuration Polarity

LL_TIM_IC_POLARITY_RISING	The circuit is sensitive to TIxFP1 rising edge, TIxFP1 is not inverted
LL_TIM_IC_POLARITY_FALLING	The circuit is sensitive to TIxFP1 falling edge, TIxFP1 is inverted
LL_TIM_IC_POLARITY_BOTHEDGE	The circuit is sensitive to both TIxFP1 rising and falling edges, TIxFP1 is not inverted

IT Defines

LL_TIM_DIER_UIE	Update interrupt enable
LL_TIM_DIER_CC1IE	Capture/compare 1 interrupt enable
LL_TIM_DIER_CC2IE	Capture/compare 2 interrupt enable

LL_TIM_DIER_CC3IE	Capture/compare 3 interrupt enable
LL_TIM_DIER_CC4IE	Capture/compare 4 interrupt enable
LL_TIM_DIER_COMIE	COM interrupt enable
LL_TIM_DIER_TIE	Trigger interrupt enable
LL_TIM_DIER_BIE	Break interrupt enable

Lock Level

LL_TIM_LOCKLEVEL_OFF	LOCK OFF - No bit is write protected
LL_TIM_LOCKLEVEL_1	LOCK Level 1
LL_TIM_LOCKLEVEL_2	LOCK Level 2
LL_TIM_LOCKLEVEL_3	LOCK Level 3

Output Configuration Idle State

LL_TIM_OCIDLESTATE_LOW	OCx=0 (after a dead-time if OC is implemented) when MOE=0
LL_TIM_OCIDLESTATE_HIGH	OCx=1 (after a dead-time if OC is implemented) when MOE=0

Output Configuration Mode

LL_TIM_OCMODE_FROZEN	The comparison between the output compare register TIMx_CCRy and the counter TIMx_CNT has no effect on the output channel level
LL_TIM_OCMODE_ACTIVE	OCyREF is forced high on compare match
LL_TIM_OCMODE_INACTIVE	OCyREF is forced low on compare match
LL_TIM_OCMODE_TOGGLE	OCyREF toggles on compare match
LL_TIM_OCMODE_FORCED_INACTIVE	OCyREF is forced low
LL_TIM_OCMODE_FORCED_ACTIVE	OCyREF is forced high
LL_TIM_OCMODE_PWM1	In upcounting, channel y is active as long as TIMx_CNT<TIMx_CCRy else inactive. In downcounting, channel y is inactive as long as TIMx_CNT>TIMx_CCRy else active.
LL_TIM_OCMODE_PWM2	In upcounting, channel y is inactive as long as TIMx_CNT<TIMx_CCRy else active. In downcounting, channel y is active as long as TIMx_CNT>TIMx_CCRy else inactive
LL_TIM_OCMODE_RETRIG_OPM1	Retrigerrable OPM mode 1
LL_TIM_OCMODE_RETRIG_OPM2	Retrigerrable OPM mode 2
LL_TIM_OCMODE_COMBINED_PWM1	Combined PWM mode 1
LL_TIM_OCMODE_COMBINED_PWM2	Combined PWM mode 2
LL_TIM_OCMODE_ASSYMETRIC_PWM1	Asymmetric PWM mode 1
LL_TIM_OCMODE_ASSYMETRIC_PWM2	Asymmetric PWM mode 2

Output Configuration Polarity

LL_TIM_OCPOLARITY_HIGH	OCxactive high
LL_TIM_OCPOLARITY_LOW	OCxactive low

OCREF clear input selection

LL_TIM_OCREF_CLR_INT_NC	OCREF_CLR_INT is not connected
LL_TIM_OCREF_CLR_INT_ETR	OCREF_CLR_INT is connected to ETRF

Output Configuration State

LL_TIM_OCSTATE_DISABLE	OCx is not active
LL_TIM_OCSTATE_ENABLE	OCx signal is output on the corresponding output pin

One Pulse Mode

LL_TIM_ONEPULSEMODE_SINGLE	Counter is not stopped at update event
LL_TIM_ONEPULSEMODE_REPETITIVE	Counter stops counting at the next update event

OSSI

LL_TIM_OSSI_DISABLE	When inactive, OCx/OCxN outputs are disabled
LL_TIM_OSSI_ENABLE	When inactive, OxC/OCxN outputs are first forced with their inactive level then forced to their idle level after the deadtime

OSSR

LL_TIM_OSSR_DISABLE	When inactive, OCx/OCxN outputs are disabled
LL_TIM_OSSR_ENABLE	When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1

Slave Mode

LL_TIM_SLAVEMODE_DISABLED	Slave mode disabled
LL_TIM_SLAVEMODE_RESET	Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter
LL_TIM_SLAVEMODE_GATED	Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high
LL_TIM_SLAVEMODE_TRIGGER	Trigger Mode - The counter starts at a rising edge of the trigger TRGI
LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER	Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter

TIM15 ENCODERMODE

LL_TIM_TIM15_ENCODERMODE_NOREDIRECTION	No redirection
LL_TIM_TIM15_ENCODERMODE_TIM2	TIM2 IC1 and TIM2 IC2 are connected to TIM15 IC1 and TIM15 IC2 respectively

LL_TIM_TIM15_ENCODERMODE_TIM3	TIM3 IC1 and TIM3 IC2 are connected to TIM15 IC1 and TIM15 IC2 respectivel y
LL_TIM_TIM15_ENCODERMODE_TIM4	TIM4 IC1 and TIM4 IC2 are connected to TIM15 IC1 and TIM15 IC2 respectively

TIM15 External Input Ch1 Remap

LL_TIM_TIM15_TI1_RMP_GPIO	TIM15 input capture 1 is connected to GPIO
LL_TIM_TIM15_TI1_RMP_LSE	TIM15 input capture 1 is connected to LSE

TIM16 External Input Ch1 Remap

LL_TIM_TIM16_TI1_RMP_GPIO	TIM16 input capture 1 is connected to GPIO
LL_TIM_TIM16_TI1_RMP_LSI	TIM16 input capture 1 is connected to LSI
LL_TIM_TIM16_TI1_RMP_LSE	TIM16 input capture 1 is connected to LSE
LL_TIM_TIM16_TI1_RMP_RTC	TIM16 input capture 1 is connected to RTC wakeup interrupt

TIM17 Timer Input Ch1 Remap

LL_TIM_TIM17_TI1_RMP_GPIO	TIM17 input capture 1 is connected to GPIO
LL_TIM_TIM17_TI1_RMP_MSI	TIM17 input capture 1 is connected to MSI
LL_TIM_TIM17_TI1_RMP_HSE_32	TIM17 input capture 1 is connected to HSE/32
LL_TIM_TIM17_TI1_RMP_MCO	TIM17 input capture 1 is connected to MCO

TIM1 External Trigger ADC1 Remap

LL_TIM_TIM1_ETR_ADC1_RMP_NC	TIM1_ETR is not connected to ADC1 analog watchdog x
LL_TIM_TIM1_ETR_ADC1_RMP_AWD1	TIM1_ETR is connected to ADC1 analog watchdog 1
LL_TIM_TIM1_ETR_ADC1_RMP_AWD2	TIM1_ETR is connected to ADC1 analog watchdog 2
LL_TIM_TIM1_ETR_ADC1_RMP_AWD3	TIM1_ETR is connected to ADC1 analog watchdog 3

TIM1 External Input Ch1 Remap

LL_TIM_TIM1_TI1_RMP_GPIO	TIM1 input capture 1 is connected to GPIO
LL_TIM_TIM1_TI1_RMP_COMP1	TIM1 input capture 1 is connected to COMP1 output

TIM2 Internal Trigger1 Remap

LL_TIM_TIM2_ITR1_RMP_TIM8_TRGO	TIM2_ITR1 is connected to TIM8_TRGO
LL_TIM_TIM2_ITR1_RMP_OTG_FS_SOF	TIM2_ITR1 is connected to OTG_FS SOF
LL_TIM_TIM2_ETR_RMP_GPIO	TIM2_ETR is connected to GPIO
LL_TIM_TIM2_ETR_RMP_LSE	TIM2_ETR is connected to LSE

TIM2 External Input Ch4 Remap

LL_TIM_TIM2_TI4_RMP_GPIO	TIM2 input capture 4 is connected to GPIO

LL_TIM_TIM2_TI4_RMP_COMP1	TIM2 input capture 4 is connected to COMP1_OUT
LL_TIM_TIM2_TI4_RMP_COMP2	TIM2 input capture 4 is connected to COMP2_OUT
LL_TIM_TIM2_TI4_RMP_COMP1_COMP2	TIM2 input capture 4 is connected to logical OR between COMP1_OUT and COMP2_OUT

TIM3 External Input Ch1 Remap

LL_TIM_TIM3_TI1_RMP_GPIO	TIM3 input capture 1 is connected to GPIO
LL_TIM_TIM3_TI1_RMP_COMP1	TIM3 input capture 1 is connected to COMP1_OUT
LL_TIM_TIM3_TI1_RMP_COMP2	TIM3 input capture 1 is connected to COMP2_OUT
LL_TIM_TIM3_TI1_RMP_COMP1_COMP2	TIM3 input capture 1 is connected to logical OR between COMP1_OUT and COMP2_OUT

TIM8 External Trigger ADC2 Remap

LL_TIM_TIM8_ETR_ADC2_RMP_NC	TIM8_ETR is not connected to ADC2 analog watchdog x
LL_TIM_TIM8_ETR_ADC2_RMP_AWD1	TIM8_ETR is connected to ADC2 analog watchdog
LL_TIM_TIM8_ETR_ADC2_RMP_AWD2	TIM8_ETR is connected to ADC2 analog watchdog 2
LL_TIM_TIM8_ETR_ADC2_RMP_AWD3	TIM8_ETR is connected to ADC2 analog watchdog 3

TIM8 External Trigger ADC3 Remap

LL_TIM_TIM8_ETR_ADC3_RMP_NC	TIM8_ETR is not connected to ADC3 analog watchdog x
LL_TIM_TIM8_ETR_ADC3_RMP_AWD1	TIM8_ETR is connected to ADC3 analog watchdog 1
LL_TIM_TIM8_ETR_ADC3_RMP_AWD2	TIM8_ETR is connected to ADC3 analog watchdog 2
LL_TIM_TIM8_ETR_ADC3_RMP_AWD3	TIM8_ETR is connected to ADC3 analog watchdog 3

TIM8 External Input Ch1 Remap

LL_TIM_TIM8_TI1_RMP_GPIO	TIM8 input capture 1 is connected to GPIO
LL_TIM_TIM8_TI1_RMP_COMP2	TIM8 input capture 1 is connected to COMP2 output

Trigger Output

LL_TIM_TRGO_RESET	UG bit from the TIMx_EGR register is used as trigger output
LL_TIM_TRGO_ENABLE	Counter Enable signal (CNT_EN) is used as trigger output
LL_TIM_TRGO_UPDATE	Update event is used as trigger output
LL_TIM_TRGO_CC1IF	CC1 capture or a compare match is used as trigger output

LL_TIM_TRGO_OC1REF	OC1REF signal is used as trigger output
LL_TIM_TRGO_OC2REF	OC2REF signal is used as trigger output
LL_TIM_TRGO_OC3REF	OC3REF signal is used as trigger output
LL_TIM_TRGO_OC4REF	OC4REF signal is used as trigger output

Trigger Output 2

LL_TIM_TRGO2_RESET	UG bit from the TIMx_EGR register is used as trigger output 2
LL_TIM_TRGO2_ENABLE	Counter Enable signal (CNT_EN) is used as trigger output 2
LL_TIM_TRGO2_UPDATE	Update event is used as trigger output 2
LL_TIM_TRGO2_CC1F	CC1 capture or a compare match is used as trigger output 2
LL_TIM_TRGO2_OC1	OC1REF signal is used as trigger output 2
LL_TIM_TRGO2_OC2	OC2REF signal is used as trigger output 2
LL_TIM_TRGO2_OC3	OC3REF signal is used as trigger output 2
LL_TIM_TRGO2_OC4	OC4REF signal is used as trigger output 2
LL_TIM_TRGO2_OC5	OC5REF signal is used as trigger output 2
LL_TIM_TRGO2_OC6	OC6REF signal is used as trigger output 2
LL_TIM_TRGO2_OC4_RISINGFALLING	OC4REF rising or falling edges are used as trigger output 2
LL_TIM_TRGO2_OC6_RISINGFALLING	OC6REF rising or falling edges are used as trigger output 2
LL_TIM_TRGO2_OC4_RISING_OC6_RISING	OC4REF or OC6REF rising edges are used as trigger output 2
LL_TIM_TRGO2_OC4_RISING_OC6_FALLING	OC4REF rising or OC6REF falling edges are used as trigger output 2
LL_TIM_TRGO2_OC5_RISING_OC6_RISING	OC5REF or OC6REF rising edges are used as trigger output 2
LL_TIM_TRGO2_OC5_RISING_OC6_FALLING	OC5REF rising or OC6REF falling edges are used as trigger output 2

Trigger Selection

LL_TIM_TS_ITR0	Internal Trigger 0 (ITR0) is used as trigger input
LL_TIM_TS_ITR1	Internal Trigger 1 (ITR1) is used as trigger input
LL_TIM_TS_ITR2	Internal Trigger 2 (ITR2) is used as trigger input
LL_TIM_TS_ITR3	Internal Trigger 3 (ITR3) is used as trigger input

LL_TIM_TS_TI1F_ED	TI1 Edge Detector (TI1F_ED) is used as trigger input
LL_TIM_TS_TI1FP1	Filtered Timer Input 1 (TI1FP1) is used as trigger input
LL_TIM_TS_TI2FP2	Filtered Timer Input 2 (TI12P2) is used as trigger input
LL_TIM_TS_ETRF	Filtered external Trigger (ETRF) is used as trigger input

Update Source

LL_TIM_UPDATESOURCE_REGULAR	Counter overflow/underflow, Setting the UG bit or Update generation through the slave mode controller generates an update request
LL_TIM_UPDATESOURCE_COUNTER	Only counter overflow/underflow generates an update request

Exported_Macros

__LL_TIM_GETFLAG_UIFCPY

Description:

- HELPER macro retrieving the UIFCPY flag from the counter value.

Parameters:

- __CNT__: Counter value

Return value:

- UIF: status bit

Notes:

- ex: __LL_TIM_GETFLAG_UIFCPY (LL_TIM_GetCounter ()); Relevant only if UIF flag remapping has been enabled (UIF status bit is copied to TIMx_CNT register bit 31)

__LL_TIM_CALC_DEADTIME

Description:

- HELPER macro calculating DTG[0:7] in the TIMx_BDTR register to achieve the requested dead time duration.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CKD__: This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4
- __DT__: deadtime duration (in ns)

Return value:

- DTG[0:7]

Notes:

- ex: __LL_TIM_CALC_DEADTIME (80000000, LL_TIM_GetClockDivision (), 120);

__LL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PSC (80000000, 1000000);

__LL_TIM_CALC_ARR

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_DELAY

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_GET_ICPSC_RATIO

Description:

- HELPER macro retrieving the ratio of the input capture prescaler.

Parameters:

- __ICPSC__: This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return value:

- Input: capture prescaler ratio (1, 2, 4 or 8)

Notes:

- ex: __LL_TIM_GET_ICPSC_RATIO (LL_TIM_IC_GetPrescaler ());

Common Write and read registers Macros

LL_TIM_WriteReg

Description:

- Write a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_TIM_ReadReg

Description:

- Read a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be read

Return value:

- Register: value

97 LL USART Generic Driver

97.1 USART Firmware driver registers structures

97.1.1 LL_USART_InitTypeDef

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***
- ***uint32_t OverSampling***

Field Documentation

- ***uint32_t LL_USART_InitTypeDef::PrescalerValue***
  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***USART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_USART_SetPrescaler()**.
- ***uint32_t LL_USART_InitTypeDef::BaudRate***
  This field defines expected Usart communication baud rate.This feature can be modified afterwards using unitary function **LL_USART_SetBaudRate()**.
- ***uint32_t LL_USART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_USART_SetDataWidth()**.
- ***uint32_t LL_USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_USART_SetStopBitsLength()**.
- ***uint32_t LL_USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_USART_SetParity()**.
- ***uint32_t LL_USART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_USART_SetTransferDirection()**.
- ***uint32_t LL_USART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_USART_SetHWFlowCtrl()**.
- ***uint32_t LL_USART_InitTypeDef::OverSampling***
  Specifies whether USART oversampling mode is 16 or 8. This parameter can be a value of ***USART_LL_EC_OVERSAMPLING***.This feature can be modified afterwards using unitary function **LL_USART_SetOverSampling()**.

97.1.2 LL_USART_ClockInitTypeDef

Data Fields

- ***uint32_t ClockOutput***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t LastBitClockPulse***

Field Documentation

- ***uint32_t LL_USART_ClockInitTypeDef::ClockOutput***
  Specifies whether the USART clock is enabled or disabled. This parameter can be a value of ***USART_LL_EC_CLOCK***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_EnableSCLKOutput()** or **LL_USART_DisableSCLKOutput()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_LL_EC_POLARITY***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPolarity()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_LL_EC_PHASE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPhase()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::LastBitClockPulse***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_LL_EC_LASTCLKPULSE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetLastClkPulseOutput()**. For more details, refer to description of this function.

97.2 USART Firmware driver API description

97.2.1 Detailed description of functions

LL_USART_Enable

Function name	**__STATIC_INLINE void LL_USART_Enable (USART_TypeDef * USARTx)**
Function description	USART Enable.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 UE LL_USART_Enable

LL_USART_Disable

Function name	**__STATIC_INLINE void LL_USART_Disable (USART_TypeDef * USARTx)**
Function description	USART Disable (all USART prescalers and outputs are disabled)
Parameters	• **USARTx:** USART Instance

Return values	• **None:**
Notes	• When USART is disabled, USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USARTx_ISR are set to their default values.
Reference Manual to LL API cross reference:	• CR1 UE LL_USART_Disable

LL_USART_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabled (USART_TypeDef * USARTx)**
Function description	Indicate if USART is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 UE LL_USART_IsEnabled

LL_USART_EnableFIFO

Function name	**__STATIC_INLINE void LL_USART_EnableFIFO (USART_TypeDef * USARTx)**
Function description	FIFO Mode Enable.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 FIFOEN LL_USART_EnableFIFO

LL_USART_DisableFIFO

Function name	**__STATIC_INLINE void LL_USART_DisableFIFO (USART_TypeDef * USARTx)**
Function description	FIFO Mode Disable.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross	• CR1 FIFOEN LL_USART_DisableFIFO

reference:

LL_USART_IsEnabledFIFO

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledFIFO (USART_TypeDef * USARTx)**
Function description	Indicate if FIFO Mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 FIFOEN LL_USART_IsEnabledFIFO

LL_USART_SetTXFIFOThreshold

Function name	**__STATIC_INLINE void LL_USART_SetTXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)**
Function description	Configure TX FIFO Threshold.
Parameters	• **USARTx:** USART Instance • **Threshold:** This parameter can be one of the following values: – LL_USART_FIFOTHRESHOLD_1_8 – LL_USART_FIFOTHRESHOLD_1_4 – LL_USART_FIFOTHRESHOLD_1_2 – LL_USART_FIFOTHRESHOLD_3_4 – LL_USART_FIFOTHRESHOLD_7_8 – LL_USART_FIFOTHRESHOLD_8_8
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 TXFTCFG LL_USART_SetTXFIFOThreshold

LL_USART_GetTXFIFOThreshold

Function name	**__STATIC_INLINE uint32_t LL_USART_GetTXFIFOThreshold (USART_TypeDef * USARTx)**
Function description	Return TX FIFO Threshold Configuration.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_FIFOTHRESHOLD_1_8 – LL_USART_FIFOTHRESHOLD_1_4

- LL_USART_FIFOTHRESHOLD_1_2
- LL_USART_FIFOTHRESHOLD_3_4
- LL_USART_FIFOTHRESHOLD_7_8
- LL_USART_FIFOTHRESHOLD_8_8

Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 TXFTCFG LL_USART_GetTXFIFOThreshold

LL_USART_SetRXFIFOThreshold

Function name	**__STATIC_INLINE void LL_USART_SetRXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)**
Function description	Configure RX FIFO Threshold.
Parameters	• **USARTx:** USART Instance • **Threshold:** This parameter can be one of the following values: – LL_USART_FIFOTHRESHOLD_1_8 – LL_USART_FIFOTHRESHOLD_1_4 – LL_USART_FIFOTHRESHOLD_1_2 – LL_USART_FIFOTHRESHOLD_3_4 – LL_USART_FIFOTHRESHOLD_7_8 – LL_USART_FIFOTHRESHOLD_8_8
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RXFTCFG LL_USART_SetRXFIFOThreshold

LL_USART_GetRXFIFOThreshold

Function name	**__STATIC_INLINE uint32_t LL_USART_GetRXFIFOThreshold (USART_TypeDef * USARTx)**
Function description	Return RX FIFO Threshold Configuration.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_FIFOTHRESHOLD_1_8 – LL_USART_FIFOTHRESHOLD_1_4 – LL_USART_FIFOTHRESHOLD_1_2 – LL_USART_FIFOTHRESHOLD_3_4 – LL_USART_FIFOTHRESHOLD_7_8 – LL_USART_FIFOTHRESHOLD_8_8
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the

	USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RXFTCFG LL_USART_GetRXFIFOThreshold

LL_USART_ConfigFIFOsThreshold

Function name	**__STATIC_INLINE void LL_USART_ConfigFIFOsThreshold (USART_TypeDef * USARTx, uint32_t TXThreshold, uint32_t RXThreshold)**
Function description	Configure TX and RX FIFOs Threshold.
Parameters	• **USARTx:** USART Instance • **TXThreshold:** This parameter can be one of the following values: – LL_USART_FIFOTHRESHOLD_1_8 – LL_USART_FIFOTHRESHOLD_1_4 – LL_USART_FIFOTHRESHOLD_1_2 – LL_USART_FIFOTHRESHOLD_3_4 – LL_USART_FIFOTHRESHOLD_7_8 – LL_USART_FIFOTHRESHOLD_8_8 • **RXThreshold:** This parameter can be one of the following values: – LL_USART_FIFOTHRESHOLD_1_8 – LL_USART_FIFOTHRESHOLD_1_4 – LL_USART_FIFOTHRESHOLD_1_2 – LL_USART_FIFOTHRESHOLD_3_4 – LL_USART_FIFOTHRESHOLD_7_8 – LL_USART_FIFOTHRESHOLD_8_8
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 TXFTCFG LL_USART_ConfigFIFOsThreshold • CR3 RXFTCFG LL_USART_ConfigFIFOsThreshold

LL_USART_EnableInStopMode

Function name	**__STATIC_INLINE void LL_USART_EnableInStopMode (USART_TypeDef * USARTx)**
Function description	USART enabled in STOP Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• When this function is enabled, USART is able to wake up the MCU from Stop mode, provided that USART clock selection is HSI or LSE in RCC. • Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode

	feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 UESM LL_USART_EnableInStopMode

LL_USART_DisableInStopMode

Function name	**__STATIC_INLINE void LL_USART_DisableInStopMode (USART_TypeDef * USARTx)**
Function description	USART disabled in STOP Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• When this function is disabled, USART is not able to wake up the MCU from Stop mode • Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 UESM LL_USART_DisableInStopMode

LL_USART_IsEnabledInStopMode

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledInStopMode (USART_TypeDef * USARTx)**
Function description	Indicate if USART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 UESM LL_USART_IsEnabledInStopMode

LL_USART_EnableDirectionRx

Function name	**__STATIC_INLINE void LL_USART_EnableDirectionRx (USART_TypeDef * USARTx)**
Function description	Receiver Enable (Receiver is enabled and begins searching for a start bit)
Parameters	• **USARTx:** USART Instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• CR1 RE LL_USART_EnableDirectionRx

LL_USART_DisableDirectionRx

Function name	**__STATIC_INLINE void LL_USART_DisableDirectionRx (USART_TypeDef * USARTx)**
Function description	Receiver Disable.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_DisableDirectionRx

LL_USART_EnableDirectionTx

Function name	**__STATIC_INLINE void LL_USART_EnableDirectionTx (USART_TypeDef * USARTx)**
Function description	Transmitter Enable.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TE LL_USART_EnableDirectionTx

LL_USART_DisableDirectionTx

Function name	**__STATIC_INLINE void LL_USART_DisableDirectionTx (USART_TypeDef * USARTx)**
Function description	Transmitter Disable.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TE LL_USART_DisableDirectionTx

LL_USART_SetTransferDirection

Function name	**__STATIC_INLINE void LL_USART_SetTransferDirection (USART_TypeDef * USARTx, uint32_t TransferDirection)**
Function description	Configure simultaneously enabled/disabled states of Transmitter and Receiver.
Parameters	• **USARTx:** USART Instance • **TransferDirection:** This parameter can be one of the following values: – LL_USART_DIRECTION_NONE

	– LL_USART_DIRECTION_RX – LL_USART_DIRECTION_TX – LL_USART_DIRECTION_TX_RX
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_SetTransferDirection • CR1 TE LL_USART_SetTransferDirection

LL_USART_GetTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_USART_GetTransferDirection (USART_TypeDef * USARTx)**
Function description	Return enabled/disabled states of Transmitter and Receiver.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_DIRECTION_NONE – LL_USART_DIRECTION_RX – LL_USART_DIRECTION_TX – LL_USART_DIRECTION_TX_RX
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_GetTransferDirection • CR1 TE LL_USART_GetTransferDirection

LL_USART_SetParity

Function name	**__STATIC_INLINE void LL_USART_SetParity (USART_TypeDef * USARTx, uint32_t Parity)**
Function description	Configure Parity (enabled/disabled and parity mode if enabled).
Parameters	• **USARTx:** USART Instance • **Parity:** This parameter can be one of the following values: – LL_USART_PARITY_NONE – LL_USART_PARITY_EVEN – LL_USART_PARITY_ODD
Return values	• **None:**
Notes	• This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (9th or 8th bit depending on data width) and parity is checked on the received data.
Reference Manual to LL API cross reference:	• CR1 PS LL_USART_SetParity • CR1 PCE LL_USART_SetParity

LL_USART_GetParity

Function name	**__STATIC_INLINE uint32_t LL_USART_GetParity (USART_TypeDef * USARTx)**

Function description	Return Parity configuration (enabled/disabled and parity mode if enabled)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_PARITY_NONE – LL_USART_PARITY_EVEN – LL_USART_PARITY_ODD
Reference Manual to LL API cross reference:	• CR1 PS LL_USART_GetParity • CR1 PCE LL_USART_GetParity

LL_USART_SetWakeUpMethod

Function name	**__STATIC_INLINE void LL_USART_SetWakeUpMethod (USART_TypeDef * USARTx, uint32_t Method)**
Function description	Set Receiver Wake Up method from Mute mode.
Parameters	• **USARTx:** USART Instance • **Method:** This parameter can be one of the following values: – LL_USART_WAKEUP_IDLELINE – LL_USART_WAKEUP_ADDRESSMARK
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 WAKE LL_USART_SetWakeUpMethod

LL_USART_GetWakeUpMethod

Function name	**__STATIC_INLINE uint32_t LL_USART_GetWakeUpMethod (USART_TypeDef * USARTx)**
Function description	Return Receiver Wake Up method from Mute mode.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_WAKEUP_IDLELINE – LL_USART_WAKEUP_ADDRESSMARK
Reference Manual to LL API cross reference:	• CR1 WAKE LL_USART_GetWakeUpMethod

LL_USART_SetDataWidth

Function name	**__STATIC_INLINE void LL_USART_SetDataWidth (USART_TypeDef * USARTx, uint32_t DataWidth)**
Function description	Set Word length (i.e.
Parameters	• **USARTx:** USART Instance • **DataWidth:** This parameter can be one of the following values: – LL_USART_DATAWIDTH_7B

- – LL_USART_DATAWIDTH_8B
- – LL_USART_DATAWIDTH_9B

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 M0 LL_USART_SetDataWidth • CR1 M1 LL_USART_SetDataWidth

LL_USART_GetDataWidth

Function name	**__STATIC_INLINE uint32_t LL_USART_GetDataWidth (USART_TypeDef * USARTx)**
Function description	Return Word length (i.e.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_DATAWIDTH_7B – LL_USART_DATAWIDTH_8B – LL_USART_DATAWIDTH_9B
Reference Manual to LL API cross reference:	• CR1 M0 LL_USART_GetDataWidth • CR1 M1 LL_USART_GetDataWidth

LL_USART_EnableMuteMode

Function name	**__STATIC_INLINE void LL_USART_EnableMuteMode (USART_TypeDef * USARTx)**
Function description	Allow switch between Mute Mode and Active mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 MME LL_USART_EnableMuteMode

LL_USART_DisableMuteMode

Function name	**__STATIC_INLINE void LL_USART_DisableMuteMode (USART_TypeDef * USARTx)**
Function description	Prevent Mute Mode use.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 MME LL_USART_DisableMuteMode

LL_USART_IsEnabledMuteMode

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledMuteMode**

	(USART_TypeDef * USARTx)
Function description	Indicate if switch between Mute Mode and Active mode is allowed.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 MME LL_USART_IsEnabledMuteMode

LL_USART_SetOverSampling

Function name	**__STATIC_INLINE void LL_USART_SetOverSampling (USART_TypeDef * USARTx, uint32_t OverSampling)**
Function description	Set Oversampling to 8-bit or 16-bit mode.
Parameters	• **USARTx:** USART Instance • **OverSampling:** This parameter can be one of the following values: – LL_USART_OVERSAMPLING_16 – LL_USART_OVERSAMPLING_8
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 OVER8 LL_USART_SetOverSampling

LL_USART_GetOverSampling

Function name	**__STATIC_INLINE uint32_t LL_USART_GetOverSampling (USART_TypeDef * USARTx)**
Function description	Return Oversampling mode.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_OVERSAMPLING_16 – LL_USART_OVERSAMPLING_8
Reference Manual to LL API cross reference:	• CR1 OVER8 LL_USART_GetOverSampling

LL_USART_SetLastClkPulseOutput

Function name	**__STATIC_INLINE void LL_USART_SetLastClkPulseOutput (USART_TypeDef * USARTx, uint32_t LastBitClockPulse)**
Function description	Configure if Clock pulse of the last data bit is output to the SCLK pin or not.
Parameters	• **USARTx:** USART Instance • **LastBitClockPulse:** This parameter can be one of the following values: – LL_USART_LASTCLKPULSE_NO_OUTPUT

	– LL_USART_LASTCLKPULSE_OUTPUT
Return values	• **None:**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBCL LL_USART_SetLastClkPulseOutput

LL_USART_GetLastClkPulseOutput

Function name	**__STATIC_INLINE uint32_t LL_USART_GetLastClkPulseOutput (USART_TypeDef * USARTx)**
Function description	Retrieve Clock pulse of the last data bit output configuration (Last bit Clock pulse output to the SCLK pin or not)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_LASTCLKPULSE_NO_OUTPUT – LL_USART_LASTCLKPULSE_OUTPUT
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBCL LL_USART_GetLastClkPulseOutput

LL_USART_SetClockPhase

Function name	**__STATIC_INLINE void LL_USART_SetClockPhase (USART_TypeDef * USARTx, uint32_t ClockPhase)**
Function description	Select the phase of the clock output on the SCLK pin in synchronous mode.
Parameters	• **USARTx:** USART Instance • **ClockPhase:** This parameter can be one of the following values: – LL_USART_PHASE_1EDGE – LL_USART_PHASE_2EDGE
Return values	• **None:**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CPHA LL_USART_SetClockPhase

LL_USART_GetClockPhase

Function name	**__STATIC_INLINE uint32_t LL_USART_GetClockPhase (USART_TypeDef * USARTx)**
Function description	Return phase of the clock output on the SCLK pin in synchronous mode.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_PHASE_1EDGE – LL_USART_PHASE_2EDGE
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CPHA LL_USART_GetClockPhase

LL_USART_SetClockPolarity

Function name	**__STATIC_INLINE void LL_USART_SetClockPolarity (USART_TypeDef * USARTx, uint32_t ClockPolarity)**
Function description	Select the polarity of the clock output on the SCLK pin in synchronous mode.
Parameters	• **USARTx:** USART Instance • **ClockPolarity:** This parameter can be one of the following values: – LL_USART_POLARITY_LOW – LL_USART_POLARITY_HIGH
Return values	• **None:**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CPOL LL_USART_SetClockPolarity

LL_USART_GetClockPolarity

Function name	**__STATIC_INLINE uint32_t LL_USART_GetClockPolarity (USART_TypeDef * USARTx)**
Function description	Return polarity of the clock output on the SCLK pin in synchronous mode.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_POLARITY_LOW – LL_USART_POLARITY_HIGH
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to

check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:	• CR2 CPOL LL_USART_GetClockPolarity

LL_USART_ConfigClock

Function name	**__STATIC_INLINE void LL_USART_ConfigClock (USART_TypeDef * USARTx, uint32_t Phase, uint32_t Polarity, uint32_t LBCPOutput)**
Function description	Configure Clock signal format (Phase Polarity and choice about output of last bit clock pulse)
Parameters	• **USARTx:** USART Instance • **Phase:** This parameter can be one of the following values: – LL_USART_PHASE_1EDGE – LL_USART_PHASE_2EDGE • **Polarity:** This parameter can be one of the following values: – LL_USART_POLARITY_LOW – LL_USART_POLARITY_HIGH • **LBCPOutput:** This parameter can be one of the following values: – LL_USART_LASTCLKPULSE_NO_OUTPUT – LL_USART_LASTCLKPULSE_OUTPUT
Return values	• **None:**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence: Clock Phase configuration using LL_USART_SetClockPhase() functionClock Polarity configuration using LL_USART_SetClockPolarity() functionOutput of Last bit Clock pulse configuration using LL_USART_SetLastClkPulseOutput() function
Reference Manual to LL API cross reference:	• CR2 CPHA LL_USART_ConfigClock • CR2 CPOL LL_USART_ConfigClock • CR2 LBCL LL_USART_ConfigClock

LL_USART_SetPrescaler

Function name	**__STATIC_INLINE void LL_USART_SetPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)**
Function description	Configure Clock source prescaler for baudrate generator and oversampling.
Parameters	• **USARTx:** USART Instance • **PrescalerValue:** This parameter can be one of the following values: – LL_USART_PRESCALER_DIV1 – LL_USART_PRESCALER_DIV2 – LL_USART_PRESCALER_DIV4

- – LL_USART_PRESCALER_DIV6
- – LL_USART_PRESCALER_DIV8
- – LL_USART_PRESCALER_DIV10
- – LL_USART_PRESCALER_DIV12
- – LL_USART_PRESCALER_DIV16
- – LL_USART_PRESCALER_DIV32
- – LL_USART_PRESCALER_DIV64
- – LL_USART_PRESCALER_DIV128
- – LL_USART_PRESCALER_DIV256

Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• PRESC PRESCALER LL_USART_SetPrescaler

LL_USART_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_USART_GetPrescaler (USART_TypeDef * USARTx)**
Function description	Retrieve the Clock source prescaler for baudrate generator and oversampling.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_PRESCALER_DIV1 – LL_USART_PRESCALER_DIV2 – LL_USART_PRESCALER_DIV4 – LL_USART_PRESCALER_DIV6 – LL_USART_PRESCALER_DIV8 – LL_USART_PRESCALER_DIV10 – LL_USART_PRESCALER_DIV12 – LL_USART_PRESCALER_DIV16 – LL_USART_PRESCALER_DIV32 – LL_USART_PRESCALER_DIV64 – LL_USART_PRESCALER_DIV128 – LL_USART_PRESCALER_DIV256
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• PRESC PRESCALER LL_USART_GetPrescaler

LL_USART_EnableSCLKOutput

Function name	**__STATIC_INLINE void LL_USART_EnableSCLKOutput (USART_TypeDef * USARTx)**

Function description	Enable Clock output on SCLK pin.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CLKEN LL_USART_EnableSCLKOutput

LL_USART_DisableSCLKOutput

Function name	**__STATIC_INLINE void LL_USART_DisableSCLKOutput (USART_TypeDef * USARTx)**
Function description	Disable Clock output on SCLK pin.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CLKEN LL_USART_DisableSCLKOutput

LL_USART_IsEnabledSCLKOutput

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledSCLKOutput (USART_TypeDef * USARTx)**
Function description	Indicate if Clock output on SCLK pin is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CLKEN LL_USART_IsEnabledSCLKOutput

LL_USART_SetStopBitsLength

Function name	**__STATIC_INLINE void LL_USART_SetStopBitsLength (USART_TypeDef * USARTx, uint32_t StopBits)**
Function description	Set the length of the stop bits.
Parameters	• **USARTx:** USART Instance • **StopBits:** This parameter can be one of the following values: – LL_USART_STOPBITS_0_5

	– LL_USART_STOPBITS_1 – LL_USART_STOPBITS_1_5 – LL_USART_STOPBITS_2
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 STOP LL_USART_SetStopBitsLength

LL_USART_GetStopBitsLength

Function name	**__STATIC_INLINE uint32_t LL_USART_GetStopBitsLength (USART_TypeDef * USARTx)**
Function description	Retrieve the length of the stop bits.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_STOPBITS_0_5 – LL_USART_STOPBITS_1 – LL_USART_STOPBITS_1_5 – LL_USART_STOPBITS_2
Reference Manual to LL API cross reference:	• CR2 STOP LL_USART_GetStopBitsLength

LL_USART_ConfigCharacter

Function name	**__STATIC_INLINE void LL_USART_ConfigCharacter (USART_TypeDef * USARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)**
Function description	Configure Character frame format (Datawidth, Parity control, Stop Bits)
Parameters	• **USARTx:** USART Instance • **DataWidth:** This parameter can be one of the following values: – LL_USART_DATAWIDTH_7B – LL_USART_DATAWIDTH_8B – LL_USART_DATAWIDTH_9B • **Parity:** This parameter can be one of the following values: – LL_USART_PARITY_NONE – LL_USART_PARITY_EVEN – LL_USART_PARITY_ODD • **StopBits:** This parameter can be one of the following values: – LL_USART_STOPBITS_0_5 – LL_USART_STOPBITS_1 – LL_USART_STOPBITS_1_5 – LL_USART_STOPBITS_2
Return values	• **None:**
Notes	• Call of this function is equivalent to following function call sequence: Data Width configuration using

	LL_USART_SetDataWidth() functionParity Control and mode configuration using LL_USART_SetParity() functionStop bits configuration using LL_USART_SetStopBitsLength() function
Reference Manual to LL API cross reference:	• CR1 PS LL_USART_ConfigCharacter • CR1 PCE LL_USART_ConfigCharacter • CR1 M0 LL_USART_ConfigCharacter • CR1 M1 LL_USART_ConfigCharacter • CR2 STOP LL_USART_ConfigCharacter

LL_USART_SetTXRXSwap

Function name	**__STATIC_INLINE void LL_USART_SetTXRXSwap (USART_TypeDef * USARTx, uint32_t SwapConfig)**
Function description	Configure TX/RX pins swapping setting.
Parameters	• **USARTx:** USART Instance • **SwapConfig:** This parameter can be one of the following values: – LL_USART_TXRX_STANDARD – LL_USART_TXRX_SWAPPED
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 SWAP LL_USART_SetTXRXSwap

LL_USART_GetTXRXSwap

Function name	**__STATIC_INLINE uint32_t LL_USART_GetTXRXSwap (USART_TypeDef * USARTx)**
Function description	Retrieve TX/RX pins swapping configuration.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_TXRX_STANDARD – LL_USART_TXRX_SWAPPED
Reference Manual to LL API cross reference:	• CR2 SWAP LL_USART_GetTXRXSwap

LL_USART_SetRXPinLevel

Function name	**__STATIC_INLINE void LL_USART_SetRXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)**
Function description	Configure RX pin active level logic.
Parameters	• **USARTx:** USART Instance • **PinInvMethod:** This parameter can be one of the following values: – LL_USART_RXPIN_LEVEL_STANDARD – LL_USART_RXPIN_LEVEL_INVERTED

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXINV LL_USART_SetRXPinLevel

LL_USART_GetRXPinLevel

Function name	**__STATIC_INLINE uint32_t LL_USART_GetRXPinLevel (USART_TypeDef * USARTx)**
Function description	Retrieve RX pin active level logic configuration.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_RXPIN_LEVEL_STANDARD – LL_USART_RXPIN_LEVEL_INVERTED
Reference Manual to LL API cross reference:	• CR2 RXINV LL_USART_GetRXPinLevel

LL_USART_SetTXPinLevel

Function name	**__STATIC_INLINE void LL_USART_SetTXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)**
Function description	Configure TX pin active level logic.
Parameters	• **USARTx:** USART Instance • **PinInvMethod:** This parameter can be one of the following values: – LL_USART_TXPIN_LEVEL_STANDARD – LL_USART_TXPIN_LEVEL_INVERTED
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXINV LL_USART_SetTXPinLevel

LL_USART_GetTXPinLevel

Function name	**__STATIC_INLINE uint32_t LL_USART_GetTXPinLevel (USART_TypeDef * USARTx)**
Function description	Retrieve TX pin active level logic configuration.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_TXPIN_LEVEL_STANDARD – LL_USART_TXPIN_LEVEL_INVERTED
Reference Manual to LL API cross reference:	• CR2 TXINV LL_USART_GetTXPinLevel

LL_USART_SetBinaryDataLogic

Function name	**__STATIC_INLINE void LL_USART_SetBinaryDataLogic (USART_TypeDef * USARTx, uint32_t DataLogic)**
Function description	Configure Binary data logic.
Parameters	• **USARTx:** USART Instance • **DataLogic:** This parameter can be one of the following values: – LL_USART_BINARY_LOGIC_POSITIVE – LL_USART_BINARY_LOGIC_NEGATIVE
Return values	• **None:**
Notes	• Allow to define how Logical data from the data register are send/received: either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)
Reference Manual to LL API cross reference:	• CR2 DATAINV LL_USART_SetBinaryDataLogic

LL_USART_GetBinaryDataLogic

Function name	**__STATIC_INLINE uint32_t LL_USART_GetBinaryDataLogic (USART_TypeDef * USARTx)**
Function description	Retrieve Binary data configuration.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_BINARY_LOGIC_POSITIVE – LL_USART_BINARY_LOGIC_NEGATIVE
Reference Manual to LL API cross reference:	• CR2 DATAINV LL_USART_GetBinaryDataLogic

LL_USART_SetTransferBitOrder

Function name	**__STATIC_INLINE void LL_USART_SetTransferBitOrder (USART_TypeDef * USARTx, uint32_t BitOrder)**
Function description	Configure transfer bit order (either Less or Most Significant Bit First)
Parameters	• **USARTx:** USART Instance • **BitOrder:** This parameter can be one of the following values: – LL_USART_BITORDER_LSBFIRST – LL_USART_BITORDER_MSBFIRST
Return values	• **None:**
Notes	• MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.
Reference Manual to LL API cross	• CR2 MSBFIRST LL_USART_SetTransferBitOrder

reference:

LL_USART_GetTransferBitOrder

Function name	**__STATIC_INLINE uint32_t LL_USART_GetTransferBitOrder (USART_TypeDef * USARTx)**
Function description	Return transfer bit order (either Less or Most Significant Bit First)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_BITORDER_LSBFIRST – LL_USART_BITORDER_MSBFIRST
Notes	• MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.
Reference Manual to LL API cross reference:	• CR2 MSBFIRST LL_USART_GetTransferBitOrder

LL_USART_EnableAutoBaudRate

Function name	**__STATIC_INLINE void LL_USART_EnableAutoBaudRate (USART_TypeDef * USARTx)**
Function description	Enable Auto Baud-Rate Detection.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 ABREN LL_USART_EnableAutoBaudRate

LL_USART_DisableAutoBaudRate

Function name	**__STATIC_INLINE void LL_USART_DisableAutoBaudRate (USART_TypeDef * USARTx)**
Function description	Disable Auto Baud-Rate Detection.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross	• CR2 ABREN LL_USART_DisableAutoBaudRate

reference:

LL_USART_IsEnabledAutoBaud

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledAutoBaud (USART_TypeDef * USARTx)**
Function description	Indicate if Auto Baud-Rate Detection mechanism is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 ABREN LL_USART_IsEnabledAutoBaud

LL_USART_SetAutoBaudRateMode

Function name	**__STATIC_INLINE void LL_USART_SetAutoBaudRateMode (USART_TypeDef * USARTx, uint32_t AutoBaudRateMode)**
Function description	Set Auto Baud-Rate mode bits.
Parameters	• **USARTx:** USART Instance • **AutoBaudRateMode:** This parameter can be one of the following values: – LL_USART_AUTOBAUD_DETECT_ON_STARTBIT – LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE – LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME – LL_USART_AUTOBAUD_DETECT_ON_55_FRAME
Return values	• **None:**
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 ABRMODE LL_USART_SetAutoBaudRateMode

LL_USART_GetAutoBaudRateMode

Function name	**__STATIC_INLINE uint32_t LL_USART_GetAutoBaudRateMode (USART_TypeDef * USARTx)**
Function description	Return Auto Baud-Rate mode.
Parameters	• **USARTx:** USART Instance

Return values	•	**Returned:** value can be one of the following values: – LL_USART_AUTOBAUD_DETECT_ON_STARTBIT – LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE – LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME – LL_USART_AUTOBAUD_DETECT_ON_55_FRAME
Notes	•	Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	•	CR2 ABRMODE LL_USART_GetAutoBaudRateMode

LL_USART_EnableRxTimeout

Function name	**__STATIC_INLINE void LL_USART_EnableRxTimeout (USART_TypeDef * USARTx)**
Function description	Enable Receiver Timeout.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RTOEN LL_USART_EnableRxTimeout

LL_USART_DisableRxTimeout

Function name	**__STATIC_INLINE void LL_USART_DisableRxTimeout (USART_TypeDef * USARTx)**
Function description	Disable Receiver Timeout.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RTOEN LL_USART_DisableRxTimeout

LL_USART_IsEnabledRxTimeout

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledRxTimeout (USART_TypeDef * USARTx)**
Function description	Indicate if Receiver Timeout feature is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RTOEN LL_USART_IsEnabledRxTimeout

LL_USART_ConfigNodeAddress

Function name	**__STATIC_INLINE void LL_USART_ConfigNodeAddress (USART_TypeDef * USARTx, uint32_t AddressLen, uint32_t NodeAddress)**
Function description	Set Address of the USART node.
Parameters	• **USARTx:** USART Instance • **AddressLen:** This parameter can be one of the following values: – LL_USART_ADDRESS_DETECT_4B – LL_USART_ADDRESS_DETECT_7B • **NodeAddress:** 4 or 7 bit Address of the USART node.
Return values	• **None:**
Notes	• This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection. • 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)
Reference Manual to LL API cross reference:	• CR2 ADD LL_USART_ConfigNodeAddress • CR2 ADDM7 LL_USART_ConfigNodeAddress

LL_USART_GetNodeAddress

Function name	**__STATIC_INLINE uint32_t LL_USART_GetNodeAddress (USART_TypeDef * USARTx)**
Function description	Return 8 bit Address of the USART node as set in ADD field of CR2.
Parameters	• **USARTx:** USART Instance
Return values	• **Address:** of the USART node (Value between Min_Data=0 and Max_Data=255)
Notes	• If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)
Reference Manual to LL API cross reference:	• CR2 ADD LL_USART_GetNodeAddress

LL_USART_GetNodeAddressLen

Function name	**__STATIC_INLINE uint32_t LL_USART_GetNodeAddressLen (USART_TypeDef * USARTx)**
Function description	Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_ADDRESS_DETECT_4B – LL_USART_ADDRESS_DETECT_7B
Reference Manual to LL API cross reference:	• CR2 ADDM7 LL_USART_GetNodeAddressLen

LL_USART_EnableRTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_EnableRTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Enable RTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RTSE LL_USART_EnableRTSHWFlowCtrl

LL_USART_DisableRTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_DisableRTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Disable RTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RTSE LL_USART_DisableRTSHWFlowCtrl

LL_USART_EnableCTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_EnableCTSHWFlowCtrl (USART_TypeDef * USARTx)**

Function description	Enable CTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSE LL_USART_EnableCTSHWFlowCtrl

LL_USART_DisableCTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_DisableCTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Disable CTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSE LL_USART_DisableCTSHWFlowCtrl

LL_USART_SetHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_SetHWFlowCtrl (USART_TypeDef * USARTx, uint32_t HardwareFlowControl)**
Function description	Configure HW Flow Control mode (both CTS and RTS)
Parameters	• **USARTx:** USART Instance • **HardwareFlowControl:** This parameter can be one of the following values: – LL_USART_HWCONTROL_NONE – LL_USART_HWCONTROL_RTS – LL_USART_HWCONTROL_CTS – LL_USART_HWCONTROL_RTS_CTS
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RTSE LL_USART_SetHWFlowCtrl • CR3 CTSE LL_USART_SetHWFlowCtrl

LL_USART_GetHWFlowCtrl

Function name	**__STATIC_INLINE uint32_t LL_USART_GetHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Return HW Flow Control configuration (both CTS and RTS)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_HWCONTROL_NONE – LL_USART_HWCONTROL_RTS – LL_USART_HWCONTROL_CTS – LL_USART_HWCONTROL_RTS_CTS
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RTSE LL_USART_GetHWFlowCtrl • CR3 CTSE LL_USART_GetHWFlowCtrl

LL_USART_EnableOneBitSamp

Function name	**__STATIC_INLINE void LL_USART_EnableOneBitSamp (USART_TypeDef * USARTx)**
Function description	Enable One bit sampling method.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 ONEBIT LL_USART_EnableOneBitSamp

LL_USART_DisableOneBitSamp

Function name	**__STATIC_INLINE void LL_USART_DisableOneBitSamp (USART_TypeDef * USARTx)**
Function description	Disable One bit sampling method.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 ONEBIT LL_USART_DisableOneBitSamp

LL_USART_IsEnabledOneBitSamp

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledOneBitSamp (USART_TypeDef * USARTx)**
Function description	Indicate if One bit sampling method is enabled.
Parameters	• **USARTx:** USART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 ONEBIT LL_USART_IsEnabledOneBitSamp

LL_USART_EnableOverrunDetect

Function name	**__STATIC_INLINE void LL_USART_EnableOverrunDetect (USART_TypeDef * USARTx)**
Function description	Enable Overrun detection.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 OVRDIS LL_USART_EnableOverrunDetect

LL_USART_DisableOverrunDetect

Function name	**__STATIC_INLINE void LL_USART_DisableOverrunDetect (USART_TypeDef * USARTx)**
Function description	Disable Overrun detection.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 OVRDIS LL_USART_DisableOverrunDetect

LL_USART_IsEnabledOverrunDetect

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledOverrunDetect (USART_TypeDef * USARTx)**
Function description	Indicate if Overrun detection is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 OVRDIS LL_USART_IsEnabledOverrunDetect

LL_USART_SetWKUPType

Function name	**__STATIC_INLINE void LL_USART_SetWKUPType (USART_TypeDef * USARTx, uint32_t Type)**
Function description	Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	• **USARTx:** USART Instance • **Type:** This parameter can be one of the following values:

	– LL_USART_WAKEUP_ON_ADDRESS – LL_USART_WAKEUP_ON_STARTBIT – LL_USART_WAKEUP_ON_RXNE
Return values	• **None:**
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 WUS LL_USART_SetWKUPType

LL_USART_GetWKUPType

Function name	**__STATIC_INLINE uint32_t LL_USART_GetWKUPType (USART_TypeDef * USARTx)**
Function description	Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_WAKEUP_ON_ADDRESS – LL_USART_WAKEUP_ON_STARTBIT – LL_USART_WAKEUP_ON_RXNE
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 WUS LL_USART_GetWKUPType

LL_USART_SetBaudRate

Function name	**__STATIC_INLINE void LL_USART_SetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling, uint32_t BaudRate)**
Function description	Configure USART BRR register for achieving expected Baud Rate value.
Parameters	• **USARTx:** USART Instance • **PeriphClk:** Peripheral Clock • **OverSampling:** This parameter can be one of the following values: – LL_USART_OVERSAMPLING_16 – LL_USART_OVERSAMPLING_8 • **BaudRate:** Baud Rate
Return values	• **None:**
Notes	• Compute and set USARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock, Oversampling mode, and expected Baud Rate values

	• Peripheral clock and Baud rate values provided as function parameters should be valid (Baud rate value != 0) • In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.
Reference Manual to LL API cross reference:	• BRR BRR LL_USART_SetBaudRate

LL_USART_GetBaudRate

Function name	**__STATIC_INLINE uint32_t LL_USART_GetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling)**
Function description	Return current Baud Rate value, according to USARTDIV present in BRR register (full BRR content), and to used Peripheral Clock and Oversampling mode values.
Parameters	• **USARTx:** USART Instance • **PeriphClk:** Peripheral Clock • **OverSampling:** This parameter can be one of the following values: – LL_USART_OVERSAMPLING_16 – LL_USART_OVERSAMPLING_8
Return values	• **Baud:** Rate
Notes	• In case of non-initialized or invalid value stored in BRR register, value 0 will be returned. • In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.
Reference Manual to LL API cross reference:	• BRR BRR LL_USART_GetBaudRate

LL_USART_SetRxTimeout

Function name	**__STATIC_INLINE void LL_USART_SetRxTimeout (USART_TypeDef * USARTx, uint32_t Timeout)**
Function description	Set Receiver Time Out Value (expressed in nb of bits duration)
Parameters	• **USARTx:** USART Instance • **Timeout:** Value between Min_Data=0x00 and Max_Data=0x00FFFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• RTOR RTO LL_USART_SetRxTimeout

LL_USART_GetRxTimeout

Function name	**__STATIC_INLINE uint32_t LL_USART_GetRxTimeout (USART_TypeDef * USARTx)**

Function description	Get Receiver Time Out Value (expressed in nb of bits duration)
Parameters	• **USARTx:** USART Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x00FFFFFF
Reference Manual to LL API cross reference:	• RTOR RTO LL_USART_GetRxTimeout

LL_USART_SetBlockLength

Function name	**__STATIC_INLINE void LL_USART_SetBlockLength (USART_TypeDef * USARTx, uint32_t BlockLength)**
Function description	Set Block Length value in reception.
Parameters	• **USARTx:** USART Instance • **BlockLength:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• RTOR BLEN LL_USART_SetBlockLength

LL_USART_GetBlockLength

Function name	**__STATIC_INLINE uint32_t LL_USART_GetBlockLength (USART_TypeDef * USARTx)**
Function description	Get Block Length value in reception.
Parameters	• **USARTx:** USART Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• RTOR BLEN LL_USART_GetBlockLength

LL_USART_EnableIrda

Function name	**__STATIC_INLINE void LL_USART_EnableIrda (USART_TypeDef * USARTx)**
Function description	Enable IrDA mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IREN LL_USART_EnableIrda

LL_USART_DisableIrda

Function name	**__STATIC_INLINE void LL_USART_DisableIrda (USART_TypeDef * USARTx)**
Function description	Disable IrDA mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IREN LL_USART_DisableIrda

LL_USART_IsEnabledIrda

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIrda (USART_TypeDef * USARTx)**
Function description	Indicate if IrDA mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IREN LL_USART_IsEnabledIrda

LL_USART_SetIrdaPowerMode

Function name	**__STATIC_INLINE void LL_USART_SetIrdaPowerMode (USART_TypeDef * USARTx, uint32_t PowerMode)**
Function description	Configure IrDA Power Mode (Normal or Low Power)
Parameters	• **USARTx:** USART Instance • **PowerMode:** This parameter can be one of the following values: – LL_USART_IRDA_POWER_NORMAL – LL_USART_IRDA_POWER_LOW
Return values	• **None:**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IRLP LL_USART_SetIrdaPowerMode

LL_USART_GetIrdaPowerMode

Function name	**__STATIC_INLINE uint32_t LL_USART_GetIrdaPowerMode (USART_TypeDef * USARTx)**
Function description	Retrieve IrDA Power Mode configuration (Normal or Low Power)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_IRDA_POWER_NORMAL – LL_USART_PHASE_2EDGE
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IRLP LL_USART_GetIrdaPowerMode

LL_USART_SetIrdaPrescaler

Function name	**__STATIC_INLINE void LL_USART_SetIrdaPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)**
Function description	Set Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)
Parameters	• **USARTx:** USART Instance • **PrescalerValue:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_SetIrdaPrescaler

LL_USART_GetIrdaPrescaler

Function name	**__STATIC_INLINE uint32_t LL_USART_GetIrdaPrescaler (USART_TypeDef * USARTx)**
Function description	Return Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)
Parameters	• **USARTx:** USART Instance
Return values	• **Irda:** prescaler value (Value between Min_Data=0x00 and Max_Data=0xFF)
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross	• GTPR PSC LL_USART_GetIrdaPrescaler

reference:

LL_USART_EnableSmartcardNACK

Function name	**__STATIC_INLINE void LL_USART_EnableSmartcardNACK (USART_TypeDef * USARTx)**
Function description	Enable Smartcard NACK transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 NACK LL_USART_EnableSmartcardNACK

LL_USART_DisableSmartcardNACK

Function name	**__STATIC_INLINE void LL_USART_DisableSmartcardNACK (USART_TypeDef * USARTx)**
Function description	Disable Smartcard NACK transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 NACK LL_USART_DisableSmartcardNACK

LL_USART_IsEnabledSmartcardNACK

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcardNACK (USART_TypeDef * USARTx)**
Function description	Indicate if Smartcard NACK transmission is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 NACK LL_USART_IsEnabledSmartcardNACK

LL_USART_EnableSmartcard

Function name	**__STATIC_INLINE void LL_USART_EnableSmartcard (USART_TypeDef * USARTx)**
Function description	Enable Smartcard mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCEN LL_USART_EnableSmartcard

LL_USART_DisableSmartcard

Function name	**__STATIC_INLINE void LL_USART_DisableSmartcard (USART_TypeDef * USARTx)**
Function description	Disable Smartcard mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCEN LL_USART_DisableSmartcard

LL_USART_IsEnabledSmartcard

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcard (USART_TypeDef * USARTx)**
Function description	Indicate if Smartcard mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCEN LL_USART_IsEnabledSmartcard

LL_USART_SetSmartcardAutoRetryCount

Function name	**__STATIC_INLINE void LL_USART_SetSmartcardAutoRetryCount (USART_TypeDef ***

	USARTx, uint32_t AutoRetryCount)
Function description	Set Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)
Parameters	• **USARTx:** USART Instance • **AutoRetryCount:** Value between Min_Data=0 and Max_Data=7
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance. • This bit-field specifies the number of retries in transmit and receive, in Smartcard mode. In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set). In reception mode, it specifies the number or erroneous reception trials, before generating a reception error (RXNE and PE bits set)
Reference Manual to LL API cross reference:	• CR3 SCARCNT LL_USART_SetSmartcardAutoRetryCount

LL_USART_GetSmartcardAutoRetryCount

Function name	**__STATIC_INLINE uint32_t LL_USART_GetSmartcardAutoRetryCount (USART_TypeDef * USARTx)**
Function description	Return Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)
Parameters	• **USARTx:** USART Instance
Return values	• **Smartcard:** Auto-Retry Count value (Value between Min_Data=0 and Max_Data=7)
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCARCNT LL_USART_GetSmartcardAutoRetryCount

LL_USART_SetSmartcardPrescaler

Function name	**__STATIC_INLINE void LL_USART_SetSmartcardPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)**
Function description	Set Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)
Parameters	• **USARTx:** USART Instance • **PrescalerValue:** Value between Min_Data=0 and Max_Data=31
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used

	to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_SetSmartcardPrescaler

LL_USART_GetSmartcardPrescaler

Function name	**__STATIC_INLINE uint32_t LL_USART_GetSmartcardPrescaler (USART_TypeDef * USARTx)**
Function description	Return Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)
Parameters	• **USARTx:** USART Instance
Return values	• **Smartcard:** prescaler value (Value between Min_Data=0 and Max_Data=31)
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_GetSmartcardPrescaler

LL_USART_SetSmartcardGuardTime

Function name	**__STATIC_INLINE void LL_USART_SetSmartcardGuardTime (USART_TypeDef * USARTx, uint32_t GuardTime)**
Function description	Set Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits: Guard time value)
Parameters	• **USARTx:** USART Instance • **GuardTime:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR GT LL_USART_SetSmartcardGuardTime

LL_USART_GetSmartcardGuardTime

Function name	**__STATIC_INLINE uint32_t LL_USART_GetSmartcardGuardTime (USART_TypeDef * USARTx)**
Function description	Return Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits: Guard time value)

Parameters	• **USARTx:** USART Instance
Return values	• **Smartcard:** Guard time value (Value between Min_Data=0x00 and Max_Data=0xFF)
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR GT LL_USART_GetSmartcardGuardTime

LL_USART_EnableHalfDuplex

Function name	**__STATIC_INLINE void LL_USART_EnableHalfDuplex (USART_TypeDef * USARTx)**
Function description	Enable Single Wire Half-Duplex mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_USART_EnableHalfDuplex

LL_USART_DisableHalfDuplex

Function name	**__STATIC_INLINE void LL_USART_DisableHalfDuplex (USART_TypeDef * USARTx)**
Function description	Disable Single Wire Half-Duplex mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_USART_DisableHalfDuplex

LL_USART_IsEnabledHalfDuplex

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex (USART_TypeDef * USARTx)**
Function description	Indicate if Single Wire Half-Duplex mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).

Notes	• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_USART_IsEnabledHalfDuplex

LL_USART_EnableSPISlave

Function name	**__STATIC_INLINE void LL_USART_EnableSPISlave (USART_TypeDef * USARTx)**
Function description	Enable SPI Synchronous Slave mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 SLVEN LL_USART_EnableSPISlave

LL_USART_DisableSPISlave

Function name	**__STATIC_INLINE void LL_USART_DisableSPISlave (USART_TypeDef * USARTx)**
Function description	Disable SPI Synchronous Slave mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 SLVEN LL_USART_DisableSPISlave

LL_USART_IsEnabledSPISlave

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlave (USART_TypeDef * USARTx)**
Function description	Indicate if SPI Synchronous Slave mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:	• CR2 SLVEN LL_USART_IsEnabledSPISlave

LL_USART_EnableSPISlaveSelect

Function name	**__STATIC_INLINE void LL_USART_EnableSPISlaveSelect (USART_TypeDef * USARTx)**
Function description	Enable SPI Slave Selection using NSS input pin.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance. • SPI Slave Selection depends on NSS input pin (The slave is selected when NSS is low and deselected when NSS is high).
Reference Manual to LL API cross reference:	• CR2 DIS_NSS LL_USART_EnableSPISlaveSelect

LL_USART_DisableSPISlaveSelect

Function name	**__STATIC_INLINE void LL_USART_DisableSPISlaveSelect (USART_TypeDef * USARTx)**
Function description	Disable SPI Slave Selection using NSS input pin.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance. • SPI Slave will be always selected and NSS input pin will be ignored.
Reference Manual to LL API cross reference:	• CR2 DIS_NSS LL_USART_DisableSPISlaveSelect

LL_USART_IsEnabledSPISlaveSelect

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlaveSelect (USART_TypeDef * USARTx)**
Function description	Indicate if SPI Slave Selection depends on NSS input pin.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is

	supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 DIS_NSS LL_USART_IsEnabledSPISlaveSelect

LL_USART_SetLINBrkDetectionLen

Function name	**__STATIC_INLINE void LL_USART_SetLINBrkDetectionLen (USART_TypeDef * USARTx, uint32_t LINBDLength)**
Function description	Set LIN Break Detection Length.
Parameters	• **USARTx:** USART Instance • **LINBDLength:** This parameter can be one of the following values: – LL_USART_LINBREAK_DETECT_10B – LL_USART_LINBREAK_DETECT_11B
Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDL LL_USART_SetLINBrkDetectionLen

LL_USART_GetLINBrkDetectionLen

Function name	**__STATIC_INLINE uint32_t LL_USART_GetLINBrkDetectionLen (USART_TypeDef * USARTx)**
Function description	Return LIN Break Detection Length.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_LINBREAK_DETECT_10B – LL_USART_LINBREAK_DETECT_11B
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDL LL_USART_GetLINBrkDetectionLen

LL_USART_EnableLIN

Function name	**__STATIC_INLINE void LL_USART_EnableLIN (USART_TypeDef * USARTx)**
Function description	Enable LIN mode.
Parameters	• **USARTx:** USART Instance

Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_EnableLIN

LL_USART_DisableLIN

Function name	**__STATIC_INLINE void LL_USART_DisableLIN (USART_TypeDef * USARTx)**
Function description	Disable LIN mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_DisableLIN

LL_USART_IsEnabledLIN

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledLIN (USART_TypeDef * USARTx)**
Function description	Indicate if LIN mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_IsEnabledLIN

LL_USART_SetDEDeassertionTime

Function name	**__STATIC_INLINE void LL_USART_SetDEDeassertionTime (USART_TypeDef * USARTx, uint32_t Time)**
Function description	Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	• **USARTx:** USART Instance • **Time:** Value between Min_Data=0 and Max_Data=31
Return values	• **None:**

Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 DEDT LL_USART_SetDEDeassertionTime

LL_USART_GetDEDeassertionTime

Function name	**__STATIC_INLINE uint32_t LL_USART_GetDEDeassertionTime (USART_TypeDef * USARTx)**
Function description	Return DEDT (Driver Enable De-Assertion Time)
Parameters	• **USARTx:** USART Instance
Return values	• **Time:** value expressed on 5 bits ([4:0] bits): Value between Min_Data=0 and Max_Data=31
Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 DEDT LL_USART_GetDEDeassertionTime

LL_USART_SetDEAssertionTime

Function name	**__STATIC_INLINE void LL_USART_SetDEAssertionTime (USART_TypeDef * USARTx, uint32_t Time)**
Function description	Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	• **USARTx:** USART Instance • **Time:** Value between Min_Data=0 and Max_Data=31
Return values	• **None:**
Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 DEAT LL_USART_SetDEAssertionTime

LL_USART_GetDEAssertionTime

Function name	**__STATIC_INLINE uint32_t LL_USART_GetDEAssertionTime (USART_TypeDef * USARTx)**
Function description	Return DEAT (Driver Enable Assertion Time)
Parameters	• **USARTx:** USART Instance
Return values	• **Time:** value expressed on 5 bits ([4:0] bits): Value between Min_Data=0 and Max_Data=31

Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 DEAT LL_USART_GetDEAssertionTime

LL_USART_EnableDEMode

Function name	**__STATIC_INLINE void LL_USART_EnableDEMode (USART_TypeDef * USARTx)**
Function description	Enable Driver Enable (DE) Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 DEM LL_USART_EnableDEMode

LL_USART_DisableDEMode

Function name	**__STATIC_INLINE void LL_USART_DisableDEMode (USART_TypeDef * USARTx)**
Function description	Disable Driver Enable (DE) Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 DEM LL_USART_DisableDEMode

LL_USART_IsEnabledDEMode

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledDEMode (USART_TypeDef * USARTx)**
Function description	Indicate if Driver Enable (DE) Mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:	• CR3 DEM LL_USART_IsEnabledDEMode

LL_USART_SetDESignalPolarity

Function name	**__STATIC_INLINE void LL_USART_SetDESignalPolarity (USART_TypeDef * USARTx, uint32_t Polarity)**
Function description	Select Driver Enable Polarity.
Parameters	• **USARTx:** USART Instance • **Polarity:** This parameter can be one of the following values: – LL_USART_DE_POLARITY_HIGH – LL_USART_DE_POLARITY_LOW
Return values	• **None:**
Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 DEP LL_USART_SetDESignalPolarity

LL_USART_GetDESignalPolarity

Function name	**__STATIC_INLINE uint32_t LL_USART_GetDESignalPolarity (USART_TypeDef * USARTx)**
Function description	Return Driver Enable Polarity.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_DE_POLARITY_HIGH – LL_USART_DE_POLARITY_LOW
Notes	• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 DEP LL_USART_GetDESignalPolarity

LL_USART_ConfigAsyncMode

Function name	**__STATIC_INLINE void LL_USART_ConfigAsyncMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Asynchronous Mode (UART)
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• In UART mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the

	USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. • Call of this function is equivalent to following function call sequence: Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function • Other remaining configurations items related to Asynchronous Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_ConfigAsyncMode • CR2 CLKEN LL_USART_ConfigAsyncMode • CR3 SCEN LL_USART_ConfigAsyncMode • CR3 IREN LL_USART_ConfigAsyncMode • CR3 HDSEL LL_USART_ConfigAsyncMode

LL_USART_ConfigSyncMode

Function name	**__STATIC_INLINE void LL_USART_ConfigSyncMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Synchronous Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• In Synchronous mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the USART in Synchronous mode. • Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence: Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() function • Other remaining configurations items related to Synchronous Mode (as Baud Rate, Word length, Parity, Clock Polarity, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_ConfigSyncMode • CR2 CLKEN LL_USART_ConfigSyncMode • CR3 SCEN LL_USART_ConfigSyncMode • CR3 IREN LL_USART_ConfigSyncMode • CR3 HDSEL LL_USART_ConfigSyncMode

LL_USART_ConfigLINMode

Function name	**__STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in LIN Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• In LIN mode, the following bits must be kept cleared: STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also set the UART/USART in LIN mode. • Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence: Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear STOP in CR2 using LL_USART_SetStopBitsLength() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet LINEN in CR2 using LL_USART_EnableLIN() function • Other remaining configurations items related to LIN Mode (as Baud Rate, Word length, LIN Break Detection Length, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 CLKEN LL_USART_ConfigLINMode • CR2 STOP LL_USART_ConfigLINMode • CR2 LINEN LL_USART_ConfigLINMode • CR3 IREN LL_USART_ConfigLINMode • CR3 SCEN LL_USART_ConfigLINMode • CR3 HDSEL LL_USART_ConfigLINMode

LL_USART_ConfigHalfDuplexMode

Function name	**__STATIC_INLINE void LL_USART_ConfigHalfDuplexMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Half Duplex Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• In Half Duplex mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register, This function also sets the UART/USART in Half Duplex mode. • Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is

supported by the USARTx instance.

- Call of this function is equivalent to following function call sequence: Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionSet HDSEL in CR3 using LL_USART_EnableHalfDuplex() function
- Other remaining configurations items related to Half Duplex Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigHalfDuplexMode
- CR2 CLKEN LL_USART_ConfigHalfDuplexMode
- CR3 HDSEL LL_USART_ConfigHalfDuplexMode
- CR3 SCEN LL_USART_ConfigHalfDuplexMode
- CR3 IREN LL_USART_ConfigHalfDuplexMode

LL_USART_ConfigSmartcardMode

Function name

__STATIC_INLINE void LL_USART_ConfigSmartcardMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Smartcard Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Smartcard mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also configures Stop bits to 1.5 bits and sets the USART in Smartcard mode (SCEN bit). Clock Output is also enabled (CLKEN).
- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence: Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() functionSet SCEN in CR3 using LL_USART_EnableSmartcard() function
- Other remaining configurations items related to Smartcard Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSmartcardMode
- CR2 STOP LL_USART_ConfigSmartcardMode
- CR2 CLKEN LL_USART_ConfigSmartcardMode
- CR3 HDSEL LL_USART_ConfigSmartcardMode

- CR3 SCEN LL_USART_ConfigSmartcardMode

LL_USART_ConfigIrdaMode

Function name	**__STATIC_INLINE void LL_USART_ConfigIrdaMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Irda Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• In IRDA mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the UART/USART in IRDA mode (IREN bit). • Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence: Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet IREN in CR3 using LL_USART_EnableIrda() function • Other remaining configurations items related to Irda Mode (as Baud Rate, Word length, Power mode, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_ConfigIrdaMode • CR2 CLKEN LL_USART_ConfigIrdaMode • CR2 STOP LL_USART_ConfigIrdaMode • CR3 SCEN LL_USART_ConfigIrdaMode • CR3 HDSEL LL_USART_ConfigIrdaMode • CR3 IREN LL_USART_ConfigIrdaMode

LL_USART_ConfigMultiProcessMode

Function name	**__STATIC_INLINE void LL_USART_ConfigMultiProcessMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Multi processor Mode (several USARTs connected in a network, one of the USARTs can be the master, its TX output connected to the RX inputs of the other slaves USARTs).
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• In MultiProcessor mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in

the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.

- Call of this function is equivalent to following function call sequence: Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Multi processor Mode (as Baud Rate, Wake Up Method, Node address, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigMultiProcessMode
- CR2 CLKEN LL_USART_ConfigMultiProcessMode
- CR3 SCEN LL_USART_ConfigMultiProcessMode
- CR3 HDSEL LL_USART_ConfigMultiProcessMode
- CR3 IREN LL_USART_ConfigMultiProcessMode

LL_USART_IsActiveFlag_PE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_PE (USART_TypeDef * USARTx)**
Function description	Check if the USART Parity Error Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR PE LL_USART_IsActiveFlag_PE

LL_USART_IsActiveFlag_FE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_FE (USART_TypeDef * USARTx)**
Function description	Check if the USART Framing Error Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR FE LL_USART_IsActiveFlag_FE

LL_USART_IsActiveFlag_NE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_NE (USART_TypeDef * USARTx)**
Function description	Check if the USART Noise error detected Flag is set or not.
Parameters	• **USARTx:** USART Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR NF LL_USART_IsActiveFlag_NE

LL_USART_IsActiveFlag_ORE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ORE (USART_TypeDef * USARTx)**
Function description	Check if the USART OverRun Error Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR ORE LL_USART_IsActiveFlag_ORE

LL_USART_IsActiveFlag_IDLE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_IDLE (USART_TypeDef * USARTx)**
Function description	Check if the USART IDLE line detected Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR IDLE LL_USART_IsActiveFlag_IDLE

LL_USART_IsActiveFlag_RXNE_RXFNE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXNE_RXFNE (USART_TypeDef * USARTx)**
Function description	Check if the USART Read Data Register or USART RX FIFO Not Empty Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR RXNE_RXFNE LL_USART_IsActiveFlag_RXNE_RXFNE

LL_USART_IsActiveFlag_TC

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TC**

	(USART_TypeDef * USARTx)
Function description	Check if the USART Transmission Complete Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TC LL_USART_IsActiveFlag_TC

LL_USART_IsActiveFlag_TXE_TXFNF

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXE_TXFNF (USART_TypeDef * USARTx)**
Function description	Check if the USART Transmit Data Register Empty or USART TX FIFO Not Full Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR TXE_TXFNF LL_USART_IsActiveFlag_TXE_TXFNF

LL_USART_IsActiveFlag_LBD

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_LBD (USART_TypeDef * USARTx)**
Function description	Check if the USART LIN Break Detection Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR LBDF LL_USART_IsActiveFlag_LBD

LL_USART_IsActiveFlag_nCTS

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_nCTS (USART_TypeDef * USARTx)**
Function description	Check if the USART CTS interrupt Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).

Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR CTSIF LL_USART_IsActiveFlag_nCTS

LL_USART_IsActiveFlag_CTS

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CTS (USART_TypeDef * USARTx)**
Function description	Check if the USART CTS Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR CTS LL_USART_IsActiveFlag_CTS

LL_USART_IsActiveFlag_RTO

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RTO (USART_TypeDef * USARTx)**
Function description	Check if the USART Receiver Time Out Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RTOF LL_USART_IsActiveFlag_RTO

LL_USART_IsActiveFlag_EOB

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_EOB (USART_TypeDef * USARTx)**
Function description	Check if the USART End Of Block Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR EOBF LL_USART_IsActiveFlag_EOB

LL_USART_IsActiveFlag_UDR

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_UDR (USART_TypeDef * USARTx)**
Function description	Check if the SPI Slave Underrun error flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR UDR LL_USART_IsActiveFlag_UDR

LL_USART_IsActiveFlag_ABRE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABRE (USART_TypeDef * USARTx)**
Function description	Check if the USART Auto-Baud Rate Error Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR ABRE LL_USART_IsActiveFlag_ABRE

LL_USART_IsActiveFlag_ABR

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABR (USART_TypeDef * USARTx)**
Function description	Check if the USART Auto-Baud Rate Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR ABRF LL_USART_IsActiveFlag_ABR

LL_USART_IsActiveFlag_BUSY

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_BUSY (USART_TypeDef * USARTx)**
Function description	Check if the USART Busy Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR BUSY LL_USART_IsActiveFlag_BUSY

LL_USART_IsActiveFlag_CM

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CM (USART_TypeDef * USARTx)**
Function description	Check if the USART Character Match Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR CMF LL_USART_IsActiveFlag_CM

LL_USART_IsActiveFlag_SBK

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_SBK (USART_TypeDef * USARTx)**
Function description	Check if the USART Send Break Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR SBKF LL_USART_IsActiveFlag_SBK

LL_USART_IsActiveFlag_RWU

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RWU (USART_TypeDef * USARTx)**
Function description	Check if the USART Receive Wake Up from mute mode Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR RWU LL_USART_IsActiveFlag_RWU

LL_USART_IsActiveFlag_WKUP

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_WKUP (USART_TypeDef * USARTx)**
Function description	Check if the USART Wake Up from stop mode Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR WUF LL_USART_IsActiveFlag_WKUP

LL_USART_IsActiveFlag_TEACK

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TEACK (USART_TypeDef * USARTx)**
Function description	Check if the USART Transmit Enable Acknowledge Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEACK LL_USART_IsActiveFlag_TEACK

LL_USART_IsActiveFlag_REACK

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_REACK (USART_TypeDef * USARTx)**
Function description	Check if the USART Receive Enable Acknowledge Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR REACK LL_USART_IsActiveFlag_REACK

LL_USART_IsActiveFlag_TXFE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFE (USART_TypeDef * USARTx)**
Function description	Check if the USART TX FIFO Empty Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).

Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR TXFE LL_USART_IsActiveFlag_TXFE

LL_USART_IsActiveFlag_RXFF

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFF (USART_TypeDef * USARTx)**
Function description	Check if the USART RX FIFO Full Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR RXFF LL_USART_IsActiveFlag_RXFF

LL_USART_IsActiveFlag_TCBGT

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TCBGT (USART_TypeDef * USARTx)**
Function description	Check if the Smartcard Transmission Complete Before Guard Time Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCBGT LL_USART_IsActiveFlag_TCBGT

LL_USART_IsActiveFlag_TXFT

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFT (USART_TypeDef * USARTx)**
Function description	Check if the USART TX FIFO Threshold Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR TXFT LL_USART_IsActiveFlag_TXFT

LL_USART_IsActiveFlag_RXFT

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFT (USART_TypeDef * USARTx)**
Function description	Check if the USART RX FIFO Threshold Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ISR RXFT LL_USART_IsActiveFlag_RXFT

LL_USART_ClearFlag_PE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_PE (USART_TypeDef * USARTx)**
Function description	Clear Parity Error Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR PECF LL_USART_ClearFlag_PE

LL_USART_ClearFlag_FE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_FE (USART_TypeDef * USARTx)**
Function description	Clear Framing Error Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR FECF LL_USART_ClearFlag_FE

LL_USART_ClearFlag_NE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_NE (USART_TypeDef * USARTx)**
Function description	Clear Noise detected Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross	• ICR NCF LL_USART_ClearFlag_NE

reference:

LL_USART_ClearFlag_ORE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_ORE (USART_TypeDef * USARTx)**
Function description	Clear OverRun Error Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR ORECF LL_USART_ClearFlag_ORE

LL_USART_ClearFlag_IDLE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_IDLE (USART_TypeDef * USARTx)**
Function description	Clear IDLE line detected Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR IDLECF LL_USART_ClearFlag_IDLE

LL_USART_ClearFlag_TXFE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_TXFE (USART_TypeDef * USARTx)**
Function description	Clear TX FIFO Empty Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ICR TXFECF LL_USART_ClearFlag_TXFE

LL_USART_ClearFlag_TC

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_TC (USART_TypeDef * USARTx)**
Function description	Clear Transmission Complete Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• ICR TCCF LL_USART_ClearFlag_TC

LL_USART_ClearFlag_TCBGT

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_TCBGT (USART_TypeDef * USARTx)**
Function description	Clear Smartcard Transmission Complete Before Guard Time Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR TCBGTCF LL_USART_ClearFlag_TCBGT

LL_USART_ClearFlag_LBD

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_LBD (USART_TypeDef * USARTx)**
Function description	Clear LIN Break Detection Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ICR LBDCF LL_USART_ClearFlag_LBD

LL_USART_ClearFlag_nCTS

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_nCTS (USART_TypeDef * USARTx)**
Function description	Clear CTS Interrupt Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ICR CTSCF LL_USART_ClearFlag_nCTS

LL_USART_ClearFlag_RTO

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_RTO**

	(USART_TypeDef * USARTx)
Function description	Clear Receiver Time Out Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• ICR RTOCF LL_USART_ClearFlag_RTO

LL_USART_ClearFlag_EOB

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_EOB (USART_TypeDef * USARTx)**
Function description	Clear End Of Block Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ICR EOBCF LL_USART_ClearFlag_EOB

LL_USART_ClearFlag_UDR

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_UDR (USART_TypeDef * USARTx)**
Function description	Clear SPI Slave Underrun Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ICR UDRCF LL_USART_ClearFlag_UDR

LL_USART_ClearFlag_CM

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_CM (USART_TypeDef * USARTx)**
Function description	Clear Character Match Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross	• ICR CMCF LL_USART_ClearFlag_CM

reference:

LL_USART_ClearFlag_WKUP

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_WKUP (USART_TypeDef * USARTx)**
Function description	Clear Wake Up from stop mode Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• ICR WUCF LL_USART_ClearFlag_WKUP

LL_USART_EnableIT_IDLE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_IDLE (USART_TypeDef * USARTx)**
Function description	Enable IDLE Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_USART_EnableIT_IDLE

LL_USART_EnableIT_RXNE_RXFNE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_RXNE_RXFNE (USART_TypeDef * USARTx)**
Function description	Enable RX Not Empty and RX FIFO Not Empty Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 RXNEIE_RXFNEIE LL_USART_EnableIT_RXNE_RXFNE

LL_USART_EnableIT_TC

Function name	**__STATIC_INLINE void LL_USART_EnableIT_TC (USART_TypeDef * USARTx)**

Function description	Enable Transmission Complete Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TCIE LL_USART_EnableIT_TC

LL_USART_EnableIT_TXE_TXFNF

Function name	**__STATIC_INLINE void LL_USART_EnableIT_TXE_TXFNF (USART_TypeDef * USARTx)**
Function description	Enable TX Empty and TX FIFO Not Full Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 TXEIE_TXFNFIE LL_USART_EnableIT_TXE_TXFNF

LL_USART_EnableIT_PE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_PE (USART_TypeDef * USARTx)**
Function description	Enable Parity Error Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 PEIE LL_USART_EnableIT_PE

LL_USART_EnableIT_CM

Function name	**__STATIC_INLINE void LL_USART_EnableIT_CM (USART_TypeDef * USARTx)**
Function description	Enable Character Match Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 CMIE LL_USART_EnableIT_CM

LL_USART_EnableIT_RTO

Function name	**__STATIC_INLINE void LL_USART_EnableIT_RTO (USART_TypeDef * USARTx)**
Function description	Enable Receiver Timeout Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RTOIE LL_USART_EnableIT_RTO

LL_USART_EnableIT_EOB

Function name	**__STATIC_INLINE void LL_USART_EnableIT_EOB (USART_TypeDef * USARTx)**
Function description	Enable End Of Block Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 EOBIE LL_USART_EnableIT_EOB

LL_USART_EnableIT_TXFE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_TXFE (USART_TypeDef * USARTx)**
Function description	Enable TX FIFO Empty Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 TXFEIE LL_USART_EnableIT_TXFE

LL_USART_EnableIT_RXFF

Function name	**__STATIC_INLINE void LL_USART_EnableIT_RXFF (USART_TypeDef * USARTx)**
Function description	Enable RX FIFO Full Interrupt.
Parameters	• **USARTx:** USART Instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RXFFIE LL_USART_EnableIT_RXFF

LL_USART_EnableIT_LBD

Function name	**__STATIC_INLINE void LL_USART_EnableIT_LBD (USART_TypeDef * USARTx)**
Function description	Enable LIN Break Detection Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDIE LL_USART_EnableIT_LBD

LL_USART_EnableIT_ERROR

Function name	**__STATIC_INLINE void LL_USART_EnableIT_ERROR (USART_TypeDef * USARTx)**
Function description	Enable Error Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.
Reference Manual to LL API cross reference:	• CR3 EIE LL_USART_EnableIT_ERROR

LL_USART_EnableIT_CTS

Function name	**__STATIC_INLINE void LL_USART_EnableIT_CTS (USART_TypeDef * USARTx)**
Function description	Enable CTS Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:	• CR3 CTSIE LL_USART_EnableIT_CTS

LL_USART_EnableIT_WKUP

Function name	**__STATIC_INLINE void LL_USART_EnableIT_WKUP (USART_TypeDef * USARTx)**
Function description	Enable Wake Up from Stop Mode Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 WUFIE LL_USART_EnableIT_WKUP

LL_USART_EnableIT_TXFT

Function name	**__STATIC_INLINE void LL_USART_EnableIT_TXFT (USART_TypeDef * USARTx)**
Function description	Enable TX FIFO Threshold Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 TXFTIE LL_USART_EnableIT_TXFT

LL_USART_EnableIT_TCBGT

Function name	**__STATIC_INLINE void LL_USART_EnableIT_TCBGT (USART_TypeDef * USARTx)**
Function description	Enable Smartcard Transmission Complete Before Guard Time Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross	• CR3 TCBGTIE LL_USART_EnableIT_TCBGT

reference:

LL_USART_EnableIT_RXFT

Function name	**__STATIC_INLINE void LL_USART_EnableIT_RXFT (USART_TypeDef * USARTx)**
Function description	Enable RX FIFO Threshold Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RXFTIE LL_USART_EnableIT_RXFT

LL_USART_DisableIT_IDLE

Function name	**__STATIC_INLINE void LL_USART_DisableIT_IDLE (USART_TypeDef * USARTx)**
Function description	Disable IDLE Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_USART_DisableIT_IDLE

LL_USART_DisableIT_RXNE_RXFNE

Function name	**__STATIC_INLINE void LL_USART_DisableIT_RXNE_RXFNE (USART_TypeDef * USARTx)**
Function description	Disable RX Not Empty and RX FIFO Not Empty Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 RXNEIE_RXFNEIE LL_USART_DisableIT_RXNE_RXFNE

LL_USART_DisableIT_TC

Function name	**__STATIC_INLINE void LL_USART_DisableIT_TC (USART_TypeDef * USARTx)**
Function description	Disable Transmission Complete Interrupt.

Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TCIE LL_USART_DisableIT_TC

LL_USART_DisableIT_TXE_TXFNF

Function name	**__STATIC_INLINE void LL_USART_DisableIT_TXE_TXFNF (USART_TypeDef * USARTx)**
Function description	Disable TX Empty and TX FIFO Not Full Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 TXEIE_TXFNFIE LL_USART_DisableIT_TXE_TXFNF

LL_USART_DisableIT_PE

Function name	**__STATIC_INLINE void LL_USART_DisableIT_PE (USART_TypeDef * USARTx)**
Function description	Disable Parity Error Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 PEIE LL_USART_DisableIT_PE

LL_USART_DisableIT_CM

Function name	**__STATIC_INLINE void LL_USART_DisableIT_CM (USART_TypeDef * USARTx)**
Function description	Disable Character Match Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 CMIE LL_USART_DisableIT_CM

LL_USART_DisableIT_RTO

Function name	**__STATIC_INLINE void LL_USART_DisableIT_RTO (USART_TypeDef * USARTx)**

Function description	Disable Receiver Timeout Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RTOIE LL_USART_DisableIT_RTO

LL_USART_DisableIT_EOB

Function name	**__STATIC_INLINE void LL_USART_DisableIT_EOB (USART_TypeDef * USARTx)**
Function description	Disable End Of Block Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 EOBIE LL_USART_DisableIT_EOB

LL_USART_DisableIT_TXFE

Function name	**__STATIC_INLINE void LL_USART_DisableIT_TXFE (USART_TypeDef * USARTx)**
Function description	Disable TX FIFO Empty Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 TXFEIE LL_USART_DisableIT_TXFE

LL_USART_DisableIT_RXFF

Function name	**__STATIC_INLINE void LL_USART_DisableIT_RXFF (USART_TypeDef * USARTx)**
Function description	Disable RX FIFO Full Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:	• CR1 RXFFIE LL_USART_DisableIT_RXFF

LL_USART_DisableIT_LBD

Function name	**__STATIC_INLINE void LL_USART_DisableIT_LBD (USART_TypeDef * USARTx)**
Function description	Disable LIN Break Detection Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDIE LL_USART_DisableIT_LBD

LL_USART_DisableIT_ERROR

Function name	**__STATIC_INLINE void LL_USART_DisableIT_ERROR (USART_TypeDef * USARTx)**
Function description	Disable Error Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.
Reference Manual to LL API cross reference:	• CR3 EIE LL_USART_DisableIT_ERROR

LL_USART_DisableIT_CTS

Function name	**__STATIC_INLINE void LL_USART_DisableIT_CTS (USART_TypeDef * USARTx)**
Function description	Disable CTS Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross	• CR3 CTSIE LL_USART_DisableIT_CTS

reference:

LL_USART_DisableIT_WKUP

Function name	**__STATIC_INLINE void LL_USART_DisableIT_WKUP (USART_TypeDef * USARTx)**
Function description	Disable Wake Up from Stop Mode Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 WUFIE LL_USART_DisableIT_WKUP

LL_USART_DisableIT_TXFT

Function name	**__STATIC_INLINE void LL_USART_DisableIT_TXFT (USART_TypeDef * USARTx)**
Function description	Disable TX FIFO Threshold Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 TXFTIE LL_USART_DisableIT_TXFT

LL_USART_DisableIT_TCBGT

Function name	**__STATIC_INLINE void LL_USART_DisableIT_TCBGT (USART_TypeDef * USARTx)**
Function description	Disable Smartcard Transmission Complete Before Guard Time Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 TCBGTIE LL_USART_DisableIT_TCBGT

LL_USART_DisableIT_RXFT

Function name	**__STATIC_INLINE void LL_USART_DisableIT_RXFT (USART_TypeDef * USARTx)**
Function description	Disable RX FIFO Threshold Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RXFTIE LL_USART_DisableIT_RXFT

LL_USART_IsEnabledIT_IDLE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_IDLE (USART_TypeDef * USARTx)**
Function description	Check if the USART IDLE Interrupt source is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_USART_IsEnabledIT_IDLE

LL_USART_IsEnabledIT_RXNE_RXFNE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXNE_RXFNE (USART_TypeDef * USARTx)**
Function description	Check if the USART RX Not Empty and USART RX FIFO Not Empty Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 RXNEIE_RXFNEIE LL_USART_IsEnabledIT_RXNE_RXFNE

LL_USART_IsEnabledIT_TC

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TC (USART_TypeDef * USARTx)**
Function description	Check if the USART Transmission Complete Interrupt is enabled or disabled.

Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TCIE LL_USART_IsEnabledIT_TC

LL_USART_IsEnabledIT_TXE_TXFNF

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE_TXFNF (USART_TypeDef * USARTx)**
Function description	Check if the USART TX Empty and USART TX FIFO Not Full Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 TXEIE_TXFNFIE LL_USART_IsEnabledIT_TXE_TXFNF

LL_USART_IsEnabledIT_PE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_PE (USART_TypeDef * USARTx)**
Function description	Check if the USART Parity Error Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 PEIE LL_USART_IsEnabledIT_PE

LL_USART_IsEnabledIT_CM

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CM (USART_TypeDef * USARTx)**
Function description	Check if the USART Character Match Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 CMIE LL_USART_IsEnabledIT_CM

LL_USART_IsEnabledIT_RTO

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RTO (USART_TypeDef * USARTx)**
Function description	Check if the USART Receiver Timeout Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RTOIE LL_USART_IsEnabledIT_RTO

LL_USART_IsEnabledIT_EOB

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_EOB (USART_TypeDef * USARTx)**
Function description	Check if the USART End Of Block Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 EOBIE LL_USART_IsEnabledIT_EOB

LL_USART_IsEnabledIT_TXFE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFE (USART_TypeDef * USARTx)**
Function description	Check if the USART TX FIFO Empty Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 TXFEIE LL_USART_IsEnabledIT_TXFE

LL_USART_IsEnabledIT_RXFF

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFF (USART_TypeDef * USARTx)**
Function description	Check if the USART RX FIFO Full Interrupt is enabled or disabled.

Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR1 RXFFIE LL_USART_IsEnabledIT_RXFF

LL_USART_IsEnabledIT_LBD

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_LBD (USART_TypeDef * USARTx)**
Function description	Check if the USART LIN Break Detection Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDIE LL_USART_IsEnabledIT_LBD

LL_USART_IsEnabledIT_ERROR

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_ERROR (USART_TypeDef * USARTx)**
Function description	Check if the USART Error Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 EIE LL_USART_IsEnabledIT_ERROR

LL_USART_IsEnabledIT_CTS

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CTS (USART_TypeDef * USARTx)**
Function description	Check if the USART CTS Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:	• CR3 CTSIE LL_USART_IsEnabledIT_CTS

LL_USART_IsEnabledIT_WKUP

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_WKUP (USART_TypeDef * USARTx)**
Function description	Check if the USART Wake Up from Stop Mode Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 WUFIE LL_USART_IsEnabledIT_WKUP

LL_USART_IsEnabledIT_TXFT

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFT (USART_TypeDef * USARTx)**
Function description	Check if USART TX FIFO Threshold Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 TXFTIE LL_USART_IsEnabledIT_TXFT

LL_USART_IsEnabledIT_TCBGT

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TCBGT (USART_TypeDef * USARTx)**
Function description	Check if the Smartcard Transmission Complete Before Guard Time Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:	• CR3 TCBGTIE LL_USART_IsEnabledIT_TCBGT

LL_USART_IsEnabledIT_RXFT

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFT (USART_TypeDef * USARTx)**
Function description	Check if USART RX FIFO Threshold Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RXFTIE LL_USART_IsEnabledIT_RXFT

LL_USART_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_USART_EnableDMAReq_RX (USART_TypeDef * USARTx)**
Function description	Enable DMA Mode for reception.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAR LL_USART_EnableDMAReq_RX

LL_USART_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_USART_DisableDMAReq_RX (USART_TypeDef * USARTx)**
Function description	Disable DMA Mode for reception.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAR LL_USART_DisableDMAReq_RX

LL_USART_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_RX (USART_TypeDef * USARTx)**
Function description	Check if DMA Mode is enabled for reception.

Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DMAR LL_USART_IsEnabledDMAReq_RX

LL_USART_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_USART_EnableDMAReq_TX (USART_TypeDef * USARTx)**
Function description	Enable DMA Mode for transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAT LL_USART_EnableDMAReq_TX

LL_USART_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_USART_DisableDMAReq_TX (USART_TypeDef * USARTx)**
Function description	Disable DMA Mode for transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAT LL_USART_DisableDMAReq_TX

LL_USART_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_TX (USART_TypeDef * USARTx)**
Function description	Check if DMA Mode is enabled for transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DMAT LL_USART_IsEnabledDMAReq_TX

LL_USART_EnableDMADeactOnRxErr

Function name	**__STATIC_INLINE void LL_USART_EnableDMADeactOnRxErr (USART_TypeDef * USARTx)**
Function description	Enable DMA Disabling on Reception Error.
Parameters	• **USARTx:** USART Instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DDRE LL_USART_EnableDMADeactOnRxErr

LL_USART_DisableDMADeactOnRxErr

Function name	**__STATIC_INLINE void LL_USART_DisableDMADeactOnRxErr (USART_TypeDef * USARTx)**
Function description	Disable DMA Disabling on Reception Error.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DDRE LL_USART_DisableDMADeactOnRxErr

LL_USART_IsEnabledDMADeactOnRxErr

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledDMADeactOnRxErr (USART_TypeDef * USARTx)**
Function description	Indicate if DMA Disabling on Reception Error is disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DDRE LL_USART_IsEnabledDMADeactOnRxErr

LL_USART_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (USART_TypeDef * USARTx, uint32_t Direction)**
Function description	Get the data register address used for DMA transfer.
Parameters	• **USARTx:** USART Instance • **Direction:** This parameter can be one of the following values: – LL_USART_DMA_REG_DATA_TRANSMIT – LL_USART_DMA_REG_DATA_RECEIVE
Return values	• **Address:** of data register
Reference Manual to LL API cross reference:	• RDR RDR LL_USART_DMA_GetRegAddr • • TDR TDR LL_USART_DMA_GetRegAddr

LL_USART_ReceiveData8

Function name	**__STATIC_INLINE uint8_t LL_USART_ReceiveData8 (USART_TypeDef * USARTx)**

Function description	Read Receiver Data register (Receive Data value, 8 bits)
Parameters	• **USARTx:** USART Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• RDR RDR LL_USART_ReceiveData8

LL_USART_ReceiveData9

Function name	**__STATIC_INLINE uint16_t LL_USART_ReceiveData9 (USART_TypeDef * USARTx)**
Function description	Read Receiver Data register (Receive Data value, 9 bits)
Parameters	• **USARTx:** USART Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x1FF
Reference Manual to LL API cross reference:	• RDR RDR LL_USART_ReceiveData9

LL_USART_TransmitData8

Function name	**__STATIC_INLINE void LL_USART_TransmitData8 (USART_TypeDef * USARTx, uint8_t Value)**
Function description	Write in Transmitter Data Register (Transmit Data value, 8 bits)
Parameters	• **USARTx:** USART Instance • **Value:** between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• TDR TDR LL_USART_TransmitData8

LL_USART_TransmitData9

Function name	**__STATIC_INLINE void LL_USART_TransmitData9 (USART_TypeDef * USARTx, uint16_t Value)**
Function description	Write in Transmitter Data Register (Transmit Data value, 9 bits)
Parameters	• **USARTx:** USART Instance • **Value:** between Min_Data=0x00 and Max_Data=0x1FF
Return values	• **None:**
Reference Manual to LL API cross reference:	• TDR TDR LL_USART_TransmitData9

LL_USART_RequestAutoBaudRate

Function name	**__STATIC_INLINE void LL_USART_RequestAutoBaudRate (USART_TypeDef * USARTx)**

Function description	Request an Automatic Baud Rate measurement on next received data frame.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• RQR ABRRQ LL_USART_RequestAutoBaudRate

LL_USART_RequestBreakSending

Function name	**__STATIC_INLINE void LL_USART_RequestBreakSending (USART_TypeDef * USARTx)**
Function description	Request Break sending.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• RQR SBKRQ LL_USART_RequestBreakSending

LL_USART_RequestEnterMuteMode

Function name	**__STATIC_INLINE void LL_USART_RequestEnterMuteMode (USART_TypeDef * USARTx)**
Function description	Put USART in mute mode and set the RWU flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• RQR MMRQ LL_USART_RequestEnterMuteMode

LL_USART_RequestRxDataFlush

Function name	**__STATIC_INLINE void LL_USART_RequestRxDataFlush (USART_TypeDef * USARTx)**
Function description	Request a Receive Data flush.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• RQR RXFRQ LL_USART_RequestRxDataFlush

LL_USART_RequestTxDataFlush

Function name	**__STATIC_INLINE void LL_USART_RequestTxDataFlush (USART_TypeDef * USARTx)**
Function description	Request a Transmit data flush.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• RQR TXFRQ LL_USART_RequestTxDataFlush

LL_USART_DeInit

Function name	**ErrorStatus LL_USART_DeInit (USART_TypeDef * USARTx)**
Function description	De-initialize USART registers (Registers restored to their default values).
Parameters	• **USARTx:** USART Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: USART registers are de-initialized – ERROR: USART registers are not de-initialized

LL_USART_Init

Function name	**ErrorStatus LL_USART_Init (USART_TypeDef * USARTx, LL_USART_InitTypeDef * USART_InitStruct)**
Function description	Initialize USART registers according to the specified parameters in USART_InitStruct.
Parameters	• **USARTx:** USART Instance • **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: USART registers are initialized according to USART_InitStruct content – ERROR: Problem occurred during USART Registers initialization
Notes	• As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned. • Baud rate value stored in USART_InitStruct BaudRate field, should be valid (different from 0).

LL_USART_StructInit

Function name	**void LL_USART_StructInit (LL_USART_InitTypeDef * USART_InitStruct)**
Function description	Set each LL_USART_InitTypeDef field to default value.
Parameters	• **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

LL_USART_ClockInit

Function name	**ErrorStatus LL_USART_ClockInit (USART_TypeDef * USARTx, LL_USART_ClockInitTypeDef * USART_ClockInitStruct)**
Function description	Initialize USART Clock related settings according to the specified parameters in the USART_ClockInitStruct.
Parameters	• **USARTx:** USART Instance • **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure that contains the Clock configuration information for the specified USART peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: USART registers related to Clock settings are initialized according to USART_ClockInitStruct content – ERROR: Problem occurred during USART Registers initialization
Notes	• As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_USART_ClockStructInit

Function name	**void LL_USART_ClockStructInit (LL_USART_ClockInitTypeDef * USART_ClockInitStruct)**
Function description	Set each field of a LL_USART_ClockInitTypeDef type structure to default value.
Parameters	• **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

97.3 USART Firmware driver defines

97.3.1 USART

Address Length Detection

LL_USART_ADDRESS_DETECT_4B	4-bit address detection method selected

LL_USART_ADDRESS_DETECT_7B	7-bit address detection (in 8-bit data mode) method selected

Autobaud Detection

LL_USART_AUTOBAUD_DETECT_ON_STARTBIT	Measurement of the start bit is used to detect the baud rate
LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE	Falling edge to falling edge measurement. Received frame must start with a single bit = 1 -> Frame = Start10xxxxxx
LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME	0x7F frame detection
LL_USART_AUTOBAUD_DETECT_ON_55_FRAME	0x55 frame detection

Binary Data Inversion

LL_USART_BINARY_LOGIC_POSITIVE	Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)
LL_USART_BINARY_LOGIC_NEGATIVE	Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_USART_BITORDER_LSBFIRST	data is transmitted/received with data bit 0 first, following the start bit
LL_USART_BITORDER_MSBFIRST	data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_USART_ICR_PECF	Parity error flag
LL_USART_ICR_FECF	Framing error flag
LL_USART_ICR_NCF	Noise detected flag
LL_USART_ICR_ORECF	Overrun error flag
LL_USART_ICR_IDLECF	Idle line detected flag
LL_USART_ICR_TXFECF	TX FIFO Empty Clear flag
LL_USART_ICR_TCCF	Transmission complete flag
LL_USART_ICR_TCBGTCF	Transmission completed before guard time flag
LL_USART_ICR_LBDCF	LIN break detection flag
LL_USART_ICR_CTSCF	CTS flag
LL_USART_ICR_RTOCF	Receiver timeout flag
LL_USART_ICR_EOBCF	End of block flag
LL_USART_ICR_UDRCF	SPI Slave Underrun Clear flag
LL_USART_ICR_CMCF	Character match flag
LL_USART_ICR_WUCF	Wakeup from Stop mode flag

Clock Signal

LL_USART_CLOCK_DISABLE	Clock signal not provided
LL_USART_CLOCK_ENABLE	Clock signal provided

Datawidth

LL_USART_DATAWIDTH_7B	7 bits word length: Start bit, 7 data bits, n stop bits
LL_USART_DATAWIDTH_8B	8 bits word length: Start bit, 8 data bits, n stop bits
LL_USART_DATAWIDTH_9B	9 bits word length: Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_USART_DE_POLARITY_HIGH	DE signal is active high
LL_USART_DE_POLARITY_LOW	DE signal is active low

Communication Direction

LL_USART_DIRECTION_NONE	Transmitter and Receiver are disabled
LL_USART_DIRECTION_RX	Transmitter is disabled and Receiver is enabled
LL_USART_DIRECTION_TX	Transmitter is enabled and Receiver is disabled
LL_USART_DIRECTION_TX_RX	Transmitter and Receiver are enabled

DMA Register Data

LL_USART_DMA_REG_DATA_TRANSMIT	Get address of data register used for transmission
LL_USART_DMA_REG_DATA_RECEIVE	Get address of data register used for reception

FIFO Threshold

LL_USART_FIFOTHRESHOLD_1_8	FIFO reaches 1/8 of its depth
LL_USART_FIFOTHRESHOLD_1_4	FIFO reaches 1/4 of its depth
LL_USART_FIFOTHRESHOLD_1_2	FIFO reaches 1/2 of its depth
LL_USART_FIFOTHRESHOLD_3_4	FIFO reaches 3/4 of its depth
LL_USART_FIFOTHRESHOLD_7_8	FIFO reaches 7/8 of its depth
LL_USART_FIFOTHRESHOLD_8_8	FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_USART_ISR_PE	Parity error flag
LL_USART_ISR_FE	Framing error flag
LL_USART_ISR_NE	Noise detected flag
LL_USART_ISR_ORE	Overrun error flag
LL_USART_ISR_IDLE	Idle line detected flag
LL_USART_ISR_RXNE_RXFNE	Read data register or RX FIFO not empty flag
LL_USART_ISR_TC	Transmission complete flag
LL_USART_ISR_TXE_TXFNF	Transmit data register empty or TX FIFO Not Full flag
LL_USART_ISR_LBDF	LIN break detection flag
LL_USART_ISR_CTSIF	CTS interrupt flag

LL_USART_ISR_CTS	CTS flag
LL_USART_ISR_RTOF	Receiver timeout flag
LL_USART_ISR_EOBF	End of block flag
LL_USART_ISR_UDR	SPI Slave underrun error flag
LL_USART_ISR_ABRE	Auto baud rate error flag
LL_USART_ISR_ABRF	Auto baud rate flag
LL_USART_ISR_BUSY	Busy flag
LL_USART_ISR_CMF	Character match flag
LL_USART_ISR_SBKF	Send break flag
LL_USART_ISR_RWU	Receiver wakeup from Mute mode flag
LL_USART_ISR_WUF	Wakeup from Stop mode flag
LL_USART_ISR_TEACK	Transmit enable acknowledge flag
LL_USART_ISR_REACK	Receive enable acknowledge flag
LL_USART_ISR_TXFE	TX FIFO empty flag
LL_USART_ISR_RXFF	RX FIFO full flag
LL_USART_ISR_TCBGT	Transmission complete before guard time completion flag
LL_USART_ISR_RXFT	RX FIFO threshold flag
LL_USART_ISR_TXFT	TX FIFO threshold flag

Hardware Control

LL_USART_HWCONTROL_NONE	CTS and RTS hardware flow control disabled
LL_USART_HWCONTROL_RTS	RTS output enabled, data is only requested when there is space in the receive buffer
LL_USART_HWCONTROL_CTS	CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)
LL_USART_HWCONTROL_RTS_CTS	CTS and RTS hardware flow control enabled

IrDA Power

LL_USART_IRDA_POWER_NORMAL	IrDA normal power mode
LL_USART_IRDA_POWER_LOW	IrDA low power mode

IT Defines

LL_USART_CR1_IDLEIE	IDLE interrupt enable
LL_USART_CR1_RXNEIE_RXFNEIE	Read data register and RXFIFO not empty interrupt enable
LL_USART_CR1_TCIE	Transmission complete interrupt enable
LL_USART_CR1_TXEIE_TXFNFIE	Transmit data register empty and TX FIFO not full interrupt enable
LL_USART_CR1_PEIE	Parity error
LL_USART_CR1_CMIE	Character match interrupt enable

LL_USART_CR1_RTOIE	Receiver timeout interrupt enable
LL_USART_CR1_EOBIE	End of Block interrupt enable
LL_USART_CR1_TXFEIE	TX FIFO empty interrupt enable
LL_USART_CR1_RXFFIE	RX FIFO full interrupt enable
LL_USART_CR2_LBDIE	LIN break detection interrupt enable
LL_USART_CR3_EIE	Error interrupt enable
LL_USART_CR3_CTSIE	CTS interrupt enable
LL_USART_CR3_WUFIE	Wakeup from Stop mode interrupt enable
LL_USART_CR3_TXFTIE	TX FIFO threshold interrupt enable
LL_USART_CR3_TCBGTIE	Transmission complete before guard time interrupt enable
LL_USART_CR3_RXFTIE	RX FIFO threshold interrupt enable

Last Clock Pulse

LL_USART_LASTCLKPULSE_NO_OUTPUT	The clock pulse of the last data bit is not output to the SCLK pin
LL_USART_LASTCLKPULSE_OUTPUT	The clock pulse of the last data bit is output to the SCLK pin

LIN Break Detection Length

LL_USART_LINBREAK_DETECT_10B	10-bit break detection method selected
LL_USART_LINBREAK_DETECT_11B	11-bit break detection method selected

Oversampling

LL_USART_OVERSAMPLING_16	Oversampling by 16
LL_USART_OVERSAMPLING_8	Oversampling by 8

Parity Control

LL_USART_PARITY_NONE	Parity control disabled
LL_USART_PARITY_EVEN	Parity control enabled and Even Parity is selected
LL_USART_PARITY_ODD	Parity control enabled and Odd Parity is selected

Clock Phase

LL_USART_PHASE_1EDGE	The first clock transition is the first data capture edge
LL_USART_PHASE_2EDGE	The second clock transition is the first data capture edge

Clock Polarity

LL_USART_POLARITY_LOW	Steady low value on SCLK pin outside transmission window
LL_USART_POLARITY_HIGH	Steady high value on SCLK pin outside transmission window

Clock Source Prescaler

LL_USART_PRESCALER_DIV1	Input clock not devided
LL_USART_PRESCALER_DIV2	Input clock devided by 2

LL_USART_PRESCALER_DIV4	Input clock devided by 4
LL_USART_PRESCALER_DIV6	Input clock devided by 6
LL_USART_PRESCALER_DIV8	Input clock devided by 8
LL_USART_PRESCALER_DIV10	Input clock devided by 10
LL_USART_PRESCALER_DIV12	Input clock devided by 12
LL_USART_PRESCALER_DIV16	Input clock devided by 16
LL_USART_PRESCALER_DIV32	Input clock devided by 32
LL_USART_PRESCALER_DIV64	Input clock devided by 64
LL_USART_PRESCALER_DIV128	Input clock devided by 128
LL_USART_PRESCALER_DIV256	Input clock devided by 256

RX Pin Active Level Inversion

LL_USART_RXPIN_LEVEL_STANDARD	RX pin signal works using the standard logic levels
LL_USART_RXPIN_LEVEL_INVERTED	RX pin signal values are inverted.

Stop Bits

LL_USART_STOPBITS_0_5	0.5 stop bit
LL_USART_STOPBITS_1	1 stop bit
LL_USART_STOPBITS_1_5	1.5 stop bits
LL_USART_STOPBITS_2	2 stop bits

TX Pin Active Level Inversion

LL_USART_TXPIN_LEVEL_STANDARD	TX pin signal works using the standard logic levels
LL_USART_TXPIN_LEVEL_INVERTED	TX pin signal values are inverted.

TX RX Pins Swap

LL_USART_TXRX_STANDARD	TX/RX pins are used as defined in standard pinout
LL_USART_TXRX_SWAPPED	TX and RX pins functions are swapped.

Wakeup

LL_USART_WAKEUP_IDLELINE	USART wake up from Mute mode on Idle Line
LL_USART_WAKEUP_ADDRESSMARK	USART wake up from Mute mode on Address Mark

Wakeup Activation

LL_USART_WAKEUP_ON_ADDRESS	Wake up active on address match
LL_USART_WAKEUP_ON_STARTBIT	Wake up active on Start bit detection
LL_USART_WAKEUP_ON_RXNE	Wake up active on RXNE

FLAG_Management

LL_USART_IsActiveFlag_RXNE

LL_USART_IsActiveFlag_TXE

IT_Management

LL_USART_EnableIT_RXNE

LL_USART_EnableIT_TXE

LL_USART_DisableIT_RXNE

LL_USART_DisableIT_TXE

LL_USART_IsEnabledIT_RXNE

LL_USART_IsEnabledIT_TXE

Exported_Macros_Helper

__LL_USART_DIV_SAMPLING8

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 8 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_8 case

__LL_USART_DIV_SAMPLING16

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 16 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_16 case

Common Write and read registers Macros

LL_USART_WriteReg

Description:

- Write a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_USART_ReadReg

Description:

- Read a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be read

Return value:

- Register: value

98 LL UTILS Generic Driver

98.1 UTILS Firmware driver registers structures

98.1.1 LL_UTILS_PLLInitTypeDef

Data Fields

- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLR***

Field Documentation

- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLM***
  Division factor for PLL VCO input clock. This parameter can be a value of ***RCC_LL_EC_PLLM_DIV***This feature can be modified afterwards using unitary function **LL_RCC_PLL_ConfigDomain_SYS()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLN***
  Multiplication factor for PLL VCO output clock. This parameter must be a number between Min_Data = 8 and Max_Data = 86This feature can be modified afterwards using unitary function **LL_RCC_PLL_ConfigDomain_SYS()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLR***
  Division for the main system clock. This parameter can be a value of ***RCC_LL_EC_PLLR_DIV***This feature can be modified afterwards using unitary function **LL_RCC_PLL_ConfigDomain_SYS()**.

98.1.2 LL_UTILS_ClkInitTypeDef

Data Fields

- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***

Field Documentation

- ***uint32_t LL_UTILS_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_LL_EC_SYSCLK_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAHBPrescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB1_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB1Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB2_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB2Prescaler()**.

98.2 UTILS Firmware driver API description

98.2.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 120000000 Hz for STM32L4Rx/STM32L4Sx devices and 80000000 Hz for others.

This section contains the following APIs:

- *LL_SetSystemCoreClock()*
- *LL_PLL_ConfigSystemClock_MSI()*
- *LL_PLL_ConfigSystemClock_HSI()*
- *LL_PLL_ConfigSystemClock_HSE()*

98.2.2 Detailed description of functions

LL_GetUID_Word0

Function name	**__STATIC_INLINE uint32_t LL_GetUID_Word0 (void)**
Function description	Get Word0 of the unique device identifier (UID based on 96 bits)
Return values	• **UID[31:0]:** X and Y coordinates on the wafer expressed in BCD format

LL_GetUID_Word1

Function name	**__STATIC_INLINE uint32_t LL_GetUID_Word1 (void)**
Function description	Get Word1 of the unique device identifier (UID based on 96 bits)
Return values	• **UID[63:32]:** Wafer number (UID[39:32]) & LOT_NUM[23:0] (UID[63:40])

LL_GetUID_Word2

Function name	**__STATIC_INLINE uint32_t LL_GetUID_Word2 (void)**
Function description	Get Word2 of the unique device identifier (UID based on 96 bits)
Return values	• **UID[95:64]:** Lot number (ASCII encoded) - LOT_NUM[55:24]

LL_GetFlashSize

Function name	**__STATIC_INLINE uint32_t LL_GetFlashSize (void)**
Function description	Get Flash memory size.
Return values	• **FLASH_SIZE[15:0]:** Flash memory size
Notes	• This bitfield indicates the size of the device Flash memory expressed in Kbytes. As an example, 0x040 corresponds to 64 Kbytes.

LL_GetPackageType

Function name	**__STATIC_INLINE uint32_t LL_GetPackageType (void)**
Function description	Get Package type.
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices. – LL_UTILS_PACKAGETYPE_LQFP64 (*) – LL_UTILS_PACKAGETYPE_LQFP100 (*)

– LL_UTILS_PACKAGETYPE_BGA132 (*)
– LL_UTILS_PACKAGETYPE_LQFP144_CSP72 (*)
– LL_UTILS_PACKAGETYPE_UFQFPN32 (*)
– LL_UTILS_PACKAGETYPE_UFQFPN48 (*)
– LL_UTILS_PACKAGETYPE_LQFP48 (*)
– LL_UTILS_PACKAGETYPE_WLCSP49 (*)
– LL_UTILS_PACKAGETYPE_UFBGA64 (*)
– LL_UTILS_PACKAGETYPE_UFBGA100 (*)
– LL_UTILS_PACKAGETYPE_UFBGA169 (*)
– LL_UTILS_PACKAGETYPE_LQFP100_DSI (*)
– LL_UTILS_PACKAGETYPE_WLCSP144_DSI (*)
– LL_UTILS_PACKAGETYPE_UFBGA144_DSI (*)
– LL_UTILS_PACKAGETYPE_UFBGA169_DSI (*)
– LL_UTILS_PACKAGETYPE_LQFP144_DSI (*)

LL_InitTick

Function name	**__STATIC_INLINE void LL_InitTick (uint32_t HCLKFrequency, uint32_t Ticks)**
Function description	This function configures the Cortex-M SysTick source of the time base.
Parameters	• **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro) • **Ticks:** Number of ticks
Return values	• **None:**
Notes	• When a RTOS is used, it is recommended to avoid changing the SysTick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_Init1msTick

Function name	**void LL_Init1msTick (uint32_t HCLKFrequency)**
Function description	This function configures the Cortex-M SysTick source to have 1ms time base.
Parameters	• **HCLKFrequency:** HCLK frequency in Hz
Return values	• **None:**
Notes	• When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service. • HCLK frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq

LL_mDelay

Function name	**void LL_mDelay (uint32_t Delay)**
Function description	This function provides accurate delay (in milliseconds) based on SysTick counter flag.
Parameters	• **Delay:** specifies the delay time length, in milliseconds.

Return values	• **None:**
Notes	• When a RTOS is used, it is recommended to avoid using blocking delay and use rather osDelay service. • To respect 1ms timebase, user should call LL_Init1msTick function which will configure Systick to 1ms

LL_SetSystemCoreClock

Function name	**void LL_SetSystemCoreClock (uint32_t HCLKFrequency)**
Function description	This function sets directly SystemCoreClock CMSIS variable.
Parameters	• **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
Return values	• **None:**
Notes	• Variable can be calculated also through SystemCoreClockUpdate function.

LL_PLL_ConfigSystemClock_MSI

Function name	**ErrorStatus LL_PLL_ConfigSystemClock_MSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)**
Function description	This function configures system clock with MSI as clock source of the PLL.
Parameters	• **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL. • **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: Max frequency configuration done – ERROR: Max frequency configuration not done
Notes	• The application needs to ensure that PLL, PLLSAI1 and/or PLLSAI2 are disabled. • Function is based on the following formula: PLL output frequency = (((MSI frequency / PLLM) * PLLN) / PLLR)PLLM: ensure that the VCO input frequency ranges from 4 to 16 MHz (PLLVCO_input = MSI frequency / PLLM)PLLN: ensure that the VCO output frequency is between 64 and 344 MHz (PLLVCO_output = PLLVCO_input * PLLN)PLLR: ensure that max frequency at 120000000 Hz is reached (PLLVCO_output / PLLR)

LL_PLL_ConfigSystemClock_HSI

Function name	**ErrorStatus LL_PLL_ConfigSystemClock_HSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)**

Function description	This function configures system clock at maximum frequency with HSI as clock source of the PLL.
Parameters	• **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL. • **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: Max frequency configuration done – ERROR: Max frequency configuration not done
Notes	• The application need to ensure that PLL, PLLSAI1 and/or PLLSAI2 are disabled. • Function is based on the following formula: PLL output frequency = (((HSI frequency / PLLM) * PLLN) / PLLR)PLLM: ensure that the VCO input frequency ranges from 4 to 16 MHz (PLLVCO_input = HSI frequency / PLLM)PLLN: ensure that the VCO output frequency is between 64 and 344 MHz (PLLVCO_output = PLLVCO_input * PLLN)PLLR: ensure that max frequency at 120000000 Hz is reach (PLLVCO_output / PLLR)

LL_PLL_ConfigSystemClock_HSE

Function name	**ErrorStatus LL_PLL_ConfigSystemClock_HSE (uint32_t HSEFrequency, uint32_t HSEBypass, LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)**
Function description	This function configures system clock with HSE as clock source of the PLL.
Parameters	• **HSEFrequency:** Value between Min_Data = 4000000 and Max_Data = 48000000 • **HSEBypass:** This parameter can be one of the following values: – LL_UTILS_HSEBYPASS_ON – LL_UTILS_HSEBYPASS_OFF • **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL. • **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: Max frequency configuration done – ERROR: Max frequency configuration not done
Notes	• The application need to ensure that PLL, PLLSAI1 and/or PLLSAI2 are disabled. • Function is based on the following formula: PLL output frequency = (((HSE frequency / PLLM) * PLLN) / PLLR)PLLM: ensure that the VCO input frequency ranges from 4 to 16 MHz (PLLVCO_input = HSE frequency / PLLM)PLLN: ensure

that the VCO output frequency is between 64 and 344 MHz (PLLVCO_output = PLLVCO_input * PLLN)PLLR: ensure that max frequency at 120000000 Hz is reached (PLLVCO_output / PLLR)

98.3 UTILS Firmware driver defines

98.3.1 UTILS

HSE Bypass activation

LL_UTILS_HSEBYPASS_OFF	HSE Bypass is not enabled
LL_UTILS_HSEBYPASS_ON	HSE Bypass is enabled

PACKAGE TYPE

LL_UTILS_PACKAGETYPE_LQFP64	LQFP64 package type
LL_UTILS_PACKAGETYPE_WLCSP64	WLCSP64 package type
LL_UTILS_PACKAGETYPE_LQFP100	LQFP100 package type
LL_UTILS_PACKAGETYPE_BGA132	BGA132 package type
LL_UTILS_PACKAGETYPE_LQFP144_CSP72	LQFP144, WLCSP81 or WLCSP72 package type
LL_UTILS_PACKAGETYPE_UFQFPN32	UFQFPN32 package type
LL_UTILS_PACKAGETYPE_UFQFPN48	UFQFPN48 package type
LL_UTILS_PACKAGETYPE_LQFP48	LQFP48 package type
LL_UTILS_PACKAGETYPE_WLCSP49	WLCSP49 package type
LL_UTILS_PACKAGETYPE_UFBGA64	UFBGA64 package type
LL_UTILS_PACKAGETYPE_UFBGA100	UFBGA100 package type
LL_UTILS_PACKAGETYPE_UFBGA169	UFBGA169 package type
LL_UTILS_PACKAGETYPE_LQFP100_DSI	LQFP100 with DSI package type
LL_UTILS_PACKAGETYPE_WLCSP144_DSI	WLCSP144 with DSI package type
LL_UTILS_PACKAGETYPE_UFBGA144_DSI	UFBGA144 with DSI package type
LL_UTILS_PACKAGETYPE_UFBGA169_DSI	UFBGA169 with DSI package type
LL_UTILS_PACKAGETYPE_LQFP144_DSI	LQFP144 with DSI package type

99 LL WWDG Generic Driver

99.1 WWDG Firmware driver API description

99.1.1 Detailed description of functions

LL_WWDG_Enable

Function name	**__STATIC_INLINE void LL_WWDG_Enable (WWDG_TypeDef * WWDGx)**
Function description	Enable Window Watchdog.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **None:**
Notes	• It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset. This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.
Reference Manual to LL API cross reference:	• CR WDGA LL_WWDG_Enable

LL_WWDG_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_WWDG_IsEnabled (WWDG_TypeDef * WWDGx)**
Function description	Checks if Window Watchdog is enabled.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR WDGA LL_WWDG_IsEnabled

LL_WWDG_SetCounter

Function name	**__STATIC_INLINE void LL_WWDG_SetCounter (WWDG_TypeDef * WWDGx, uint32_t Counter)**
Function description	Set the Watchdog counter value to provided value (7-bits T[6:0])
Parameters	• **WWDGx:** WWDG Instance • **Counter:** 0..0x7F (7 bit counter value)
Return values	• **None:**
Notes	• When writing to the WWDG_CR register, always write 1 in the MSB b6 to avoid generating an immediate reset This counter is decremented every (4096 x 2expWDGTB) PCLK cycles A reset is produced when it rolls over from 0x40 to 0x3F (bit T6

	becomes cleared) Setting the counter lower then 0x40 causes an immediate reset (if WWDG enabled)
Reference Manual to LL API cross reference:	• CR T LL_WWDG_SetCounter

LL_WWDG_GetCounter

Function name	**__STATIC_INLINE uint32_t LL_WWDG_GetCounter (WWDG_TypeDef * WWDGx)**
Function description	Return current Watchdog Counter Value (7 bits counter value)
Parameters	• **WWDGx:** WWDG Instance
Return values	• **7:** bit Watchdog Counter value
Reference Manual to LL API cross reference:	• CR T LL_WWDG_GetCounter

LL_WWDG_SetPrescaler

Function name	**__STATIC_INLINE void LL_WWDG_SetPrescaler (WWDG_TypeDef * WWDGx, uint32_t Prescaler)**
Function description	Set the time base of the prescaler (WDGTB).
Parameters	• **WWDGx:** WWDG Instance • **Prescaler:** This parameter can be one of the following values: – LL_WWDG_PRESCALER_1 – LL_WWDG_PRESCALER_2 – LL_WWDG_PRESCALER_4 – LL_WWDG_PRESCALER_8
Return values	• **None:**
Notes	• Prescaler is used to apply ratio on PCLK clock, so that Watchdog counter is decremented every (4096 x 2expWDGTB) PCLK cycles
Reference Manual to LL API cross reference:	• CFR WDGTB LL_WWDG_SetPrescaler

LL_WWDG_GetPrescaler

Function name	**__STATIC_INLINE uint32_t LL_WWDG_GetPrescaler (WWDG_TypeDef * WWDGx)**
Function description	Return current Watchdog Prescaler Value.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **Returned:** value can be one of the following values: – LL_WWDG_PRESCALER_1 – LL_WWDG_PRESCALER_2 – LL_WWDG_PRESCALER_4

– LL_WWDG_PRESCALER_8

Reference Manual to LL API cross reference:
- CFR WDGTB LL_WWDG_GetPrescaler

LL_WWDG_SetWindow

Function name
__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)

Function description
Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).

Parameters
- **WWDGx:** WWDG Instance
- **Window:** 0x00..0x7F (7 bit Window value)

Return values
- **None:**

Notes
- This window value defines when write in the WWDG_CR register to program Watchdog counter is allowed. Watchdog counter value update must occur only when the counter value is lower than the Watchdog window register value. Otherwise, a MCU reset is generated if the 7-bit Watchdog counter value (in the control register) is refreshed before the downcounter has reached the watchdog window register value. Physically is possible to set the Window lower then 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.

Reference Manual to LL API cross reference:
- CFR W LL_WWDG_SetWindow

LL_WWDG_GetWindow

Function name
__STATIC_INLINE uint32_t LL_WWDG_GetWindow (WWDG_TypeDef * WWDGx)

Function description
Return current Watchdog Window Value (7 bits value)

Parameters
- **WWDGx:** WWDG Instance

Return values
- **7:** bit Watchdog Window value

Reference Manual to LL API cross reference:
- CFR W LL_WWDG_GetWindow

LL_WWDG_IsActiveFlag_EWKUP

Function name
__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (WWDG_TypeDef * WWDGx)

Function description
Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.

Parameters
- **WWDGx:** WWDG Instance

Return values
- **State:** of bit (1 or 0).

Notes
- This bit is set by hardware when the counter has reached the

	value 0x40. It must be cleared by software by writing 0. A write of 1 has no effect. This bit is also set if the interrupt is not enabled.
Reference Manual to LL API cross reference:	• SR EWIF LL_WWDG_IsActiveFlag_EWKUP

LL_WWDG_ClearFlag_EWKUP

Function name	**__STATIC_INLINE void LL_WWDG_ClearFlag_EWKUP (WWDG_TypeDef * WWDGx)**
Function description	Clear WWDG Early Wakeup Interrupt Flag (EWIF)
Parameters	• **WWDGx:** WWDG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR EWIF LL_WWDG_ClearFlag_EWKUP

LL_WWDG_EnableIT_EWKUP

Function name	**__STATIC_INLINE void LL_WWDG_EnableIT_EWKUP (WWDG_TypeDef * WWDGx)**
Function description	Enable the Early Wakeup Interrupt.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **None:**
Notes	• When set, an interrupt occurs whenever the counter reaches value 0x40. This interrupt is only cleared by hardware after a reset
Reference Manual to LL API cross reference:	• CFR EWI LL_WWDG_EnableIT_EWKUP

LL_WWDG_IsEnabledIT_EWKUP

Function name	**__STATIC_INLINE uint32_t LL_WWDG_IsEnabledIT_EWKUP (WWDG_TypeDef * WWDGx)**
Function description	Check if Early Wakeup Interrupt is enabled.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CFR EWI LL_WWDG_IsEnabledIT_EWKUP

99.2 WWDG Firmware driver defines

99.2.1 WWDG

IT Defines

LL_WWDG_CFR_EWI

PRESCALER

LL_WWDG_PRESCALER_1	WWDG counter clock = (PCLK1/4096)/1
LL_WWDG_PRESCALER_2	WWDG counter clock = (PCLK1/4096)/2
LL_WWDG_PRESCALER_4	WWDG counter clock = (PCLK1/4096)/4
LL_WWDG_PRESCALER_8	WWDG counter clock = (PCLK1/4096)/8

Common Write and read registers macros

LL_WWDG_WriteReg

Description:

- Write a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_WWDG_ReadReg

Description:

- Read a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

100 Correspondence between API registers and API low-layer driver functions

100.1 ADC

Table 26: Correspondence between ADC registers and ADC low-layer driver functions

Register	Field	Function
AWD2CR	AWD2CH	*LL_ADC_GetAnalogWDMonitChannels*
		LL_ADC_SetAnalogWDMonitChannels
AWD3CR	AWD3CH	*LL_ADC_GetAnalogWDMonitChannels*
		LL_ADC_SetAnalogWDMonitChannels
CALFACT	CALFACT_D	*LL_ADC_GetCalibrationFactor*
		LL_ADC_SetCalibrationFactor
	CALFACT_S	*LL_ADC_GetCalibrationFactor*
		LL_ADC_SetCalibrationFactor
CCR	CKMODE	*LL_ADC_GetCommonClock*
		LL_ADC_SetCommonClock
	PRESC	*LL_ADC_GetCommonClock*
		LL_ADC_SetCommonClock
	TSEN	*LL_ADC_GetCommonPathInternalCh*
		LL_ADC_SetCommonPathInternalCh
	VBATEN	*LL_ADC_GetCommonPathInternalCh*
		LL_ADC_SetCommonPathInternalCh
	VREFEN	*LL_ADC_GetCommonPathInternalCh*
		LL_ADC_SetCommonPathInternalCh
CDR	RDATA_MST	*LL_ADC_DMA_GetRegAddr*
	RDATA_SLV	*LL_ADC_DMA_GetRegAddr*
CFGR	ALIGN	*LL_ADC_GetDataAlignment*
		LL_ADC_SetDataAlignment
	AUTDLY	*LL_ADC_GetLowPowerMode*
		LL_ADC_SetLowPowerMode
	AWD1CH	*LL_ADC_GetAnalogWDMonitChannels*
		LL_ADC_SetAnalogWDMonitChannels
	AWD1EN	*LL_ADC_GetAnalogWDMonitChannels*
		LL_ADC_SetAnalogWDMonitChannels
	AWD1SGL	*LL_ADC_GetAnalogWDMonitChannels*
		LL_ADC_SetAnalogWDMonitChannels

Register	Field	Function
	CONT	*LL_ADC_REG_GetContinuousMode*
		LL_ADC_REG_SetContinuousMode
	DFSDMCFG	*LL_ADC_REG_GetDFSDMTransfer*
	DISCEN	*LL_ADC_REG_GetSequencerDiscont*
		LL_ADC_REG_SetSequencerDiscont
	DISCNUM	*LL_ADC_REG_GetSequencerDiscont*
		LL_ADC_REG_SetSequencerDiscont
	DMACFG	*LL_ADC_REG_GetDMATransfer*
		LL_ADC_REG_SetDMATransfer
	DMAEN	*LL_ADC_REG_GetDMATransfer*
		LL_ADC_REG_SetDMATransfer
	EXTEN	*LL_ADC_REG_GetTriggerEdge*
		LL_ADC_REG_GetTriggerSource
		LL_ADC_REG_IsTriggerSourceSWStart
		LL_ADC_REG_SetTriggerEdge
		LL_ADC_REG_SetTriggerSource
	EXTSEL	*LL_ADC_REG_GetTriggerSource*
		LL_ADC_REG_SetTriggerSource
	JAUTO	*LL_ADC_INJ_GetTrigAuto*
		LL_ADC_INJ_SetTrigAuto
	JAWD1EN	*LL_ADC_GetAnalogWDMonitChannels*
		LL_ADC_SetAnalogWDMonitChannels
	JDISCEN	*LL_ADC_INJ_GetSequencerDiscont*
		LL_ADC_INJ_SetSequencerDiscont
	JQDIS	*LL_ADC_INJ_GetQueueMode*
		LL_ADC_INJ_SetQueueMode
	JQM	*LL_ADC_INJ_GetQueueMode*
		LL_ADC_INJ_SetQueueMode
	OVRMOD	*LL_ADC_REG_GetOverrun*
		LL_ADC_REG_SetOverrun
	RES	*LL_ADC_GetResolution*
		LL_ADC_SetResolution
CFGR2	JOVSE	*LL_ADC_GetOverSamplingScope*
		LL_ADC_SetOverSamplingScope
	OVSR	*LL_ADC_ConfigOverSamplingRatioShift*
		LL_ADC_GetOverSamplingRatio

Register	Field	Function
	OVSS	LL_ADC_ConfigOverSamplingRatioShift
		LL_ADC_GetOverSamplingShift
	ROVSE	LL_ADC_GetOverSamplingScope
		LL_ADC_SetOverSamplingScope
	ROVSM	LL_ADC_GetOverSamplingScope
		LL_ADC_SetOverSamplingScope
	TROVS	LL_ADC_GetOverSamplingDiscont
		LL_ADC_SetOverSamplingDiscont
CR	ADCAL	LL_ADC_IsCalibrationOnGoing
		LL_ADC_StartCalibration
	ADCALDIF	LL_ADC_StartCalibration
	ADDIS	LL_ADC_Disable
		LL_ADC_IsDisableOngoing
	ADEN	LL_ADC_Enable
		LL_ADC_IsEnabled
	ADSTART	LL_ADC_REG_IsConversionOngoing
		LL_ADC_REG_StartConversion
	ADSTP	LL_ADC_REG_IsStopConversionOngoing
		LL_ADC_REG_StopConversion
	ADVREGEN	LL_ADC_DisableInternalRegulator
		LL_ADC_EnableInternalRegulator
		LL_ADC_IsInternalRegulatorEnabled
	DEEPPWD	LL_ADC_DisableDeepPowerDown
		LL_ADC_EnableDeepPowerDown
		LL_ADC_IsDeepPowerDownEnabled
	JADSTART	LL_ADC_INJ_IsConversionOngoing
		LL_ADC_INJ_StartConversion
	JADSTP	LL_ADC_INJ_IsStopConversionOngoing
		LL_ADC_INJ_StopConversion
DIFSEL	DIFSEL	LL_ADC_GetChannelSamplingTime
DR	RDATA	LL_ADC_DMA_GetRegAddr
		LL_ADC_REG_ReadConversionData10
		LL_ADC_REG_ReadConversionData12
		LL_ADC_REG_ReadConversionData32
		LL_ADC_REG_ReadConversionData6
		LL_ADC_REG_ReadConversionData8

Register	Field	Function
IER	ADRDYIE	LL_ADC_DisableIT_ADRDY
		LL_ADC_EnableIT_ADRDY
		LL_ADC_IsEnabledIT_ADRDY
	AWD1IE	LL_ADC_DisableIT_AWD1
		LL_ADC_EnableIT_AWD1
		LL_ADC_IsEnabledIT_AWD1
	AWD2IE	LL_ADC_DisableIT_AWD2
		LL_ADC_EnableIT_AWD2
		LL_ADC_IsEnabledIT_AWD2
	AWD3IE	LL_ADC_DisableIT_AWD3
		LL_ADC_EnableIT_AWD3
		LL_ADC_IsEnabledIT_AWD3
	EOCIE	LL_ADC_DisableIT_EOC
		LL_ADC_EnableIT_EOC
		LL_ADC_IsEnabledIT_EOC
	EOSIE	LL_ADC_DisableIT_EOS
		LL_ADC_EnableIT_EOS
		LL_ADC_IsEnabledIT_EOS
	EOSMPIE	LL_ADC_DisableIT_EOSMP
		LL_ADC_EnableIT_EOSMP
		LL_ADC_IsEnabledIT_EOSMP
	JEOCIE	LL_ADC_DisableIT_JEOC
		LL_ADC_EnableIT_JEOC
		LL_ADC_IsEnabledIT_JEOC
	JEOSIE	LL_ADC_DisableIT_JEOS
		LL_ADC_EnableIT_JEOS
		LL_ADC_IsEnabledIT_JEOS
	JQOVFIE	LL_ADC_DisableIT_JQOVF
		LL_ADC_EnableIT_JQOVF
		LL_ADC_IsEnabledIT_JQOVF
	OVRIE	LL_ADC_DisableIT_OVR
		LL_ADC_EnableIT_OVR
		LL_ADC_IsEnabledIT_OVR
ISR	ADRDY	LL_ADC_ClearFlag_ADRDY
		LL_ADC_IsActiveFlag_ADRDY
	AWD1	LL_ADC_ClearFlag_AWD1

Register	Field	Function
		LL_ADC_IsActiveFlag_AWD1
	AWD2	LL_ADC_ClearFlag_AWD2
		LL_ADC_IsActiveFlag_AWD2
	AWD3	LL_ADC_ClearFlag_AWD3
		LL_ADC_IsActiveFlag_AWD3
	EOC	LL_ADC_ClearFlag_EOC
		LL_ADC_IsActiveFlag_EOC
	EOS	LL_ADC_ClearFlag_EOS
		LL_ADC_IsActiveFlag_EOS
	EOSMP	LL_ADC_ClearFlag_EOSMP
		LL_ADC_IsActiveFlag_EOSMP
	JEOC	LL_ADC_ClearFlag_JEOC
		LL_ADC_IsActiveFlag_JEOC
	JEOS	LL_ADC_ClearFlag_JEOS
		LL_ADC_IsActiveFlag_JEOS
	JQOVF	LL_ADC_ClearFlag_JQOVF
		LL_ADC_IsActiveFlag_JQOVF
	OVR	LL_ADC_ClearFlag_OVR
		LL_ADC_IsActiveFlag_OVR
JDR1	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData6
		LL_ADC_INJ_ReadConversionData8
JDR2	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData6
		LL_ADC_INJ_ReadConversionData8
JDR3	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData6
		LL_ADC_INJ_ReadConversionData8
JDR4	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12

Register	Field	Function
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData6
		LL_ADC_INJ_ReadConversionData8
JSQR	JEXTEN	*LL_ADC_INJ_ConfigQueueContext*
		LL_ADC_INJ_GetTriggerEdge
		LL_ADC_INJ_GetTriggerSource
		LL_ADC_INJ_IsTriggerSourceSWStart
		LL_ADC_INJ_SetTriggerEdge
		LL_ADC_INJ_SetTriggerSource
	JEXTSEL	*LL_ADC_INJ_ConfigQueueContext*
		LL_ADC_INJ_GetTriggerSource
		LL_ADC_INJ_SetTriggerSource
	JL	*LL_ADC_INJ_ConfigQueueContext*
		LL_ADC_INJ_GetSequencerLength
		LL_ADC_INJ_SetSequencerLength
	JSQ1	*LL_ADC_INJ_ConfigQueueContext*
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
	JSQ2	*LL_ADC_INJ_ConfigQueueContext*
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
	JSQ3	*LL_ADC_INJ_ConfigQueueContext*
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
	JSQ4	*LL_ADC_INJ_ConfigQueueContext*
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
OFR1	OFFSET1	*LL_ADC_GetOffsetLevel*
		LL_ADC_SetOffset
	OFFSET1_CH	*LL_ADC_GetOffsetChannel*
		LL_ADC_SetOffset
	OFFSET1_EN	*LL_ADC_GetOffsetState*
		LL_ADC_SetOffset
		LL_ADC_SetOffsetState
OFR2	OFFSET2	*LL_ADC_GetOffsetLevel*
		LL_ADC_SetOffset

Register	Field	Function
	OFFSET2_CH	*LL_ADC_GetOffsetChannel*
		LL_ADC_SetOffset
	OFFSET2_EN	*LL_ADC_GetOffsetState*
		LL_ADC_SetOffset
		LL_ADC_SetOffsetState
OFR3	OFFSET3	*LL_ADC_GetOffsetLevel*
		LL_ADC_SetOffset
	OFFSET3_CH	*LL_ADC_GetOffsetChannel*
		LL_ADC_SetOffset
	OFFSET3_EN	*LL_ADC_GetOffsetState*
		LL_ADC_SetOffset
		LL_ADC_SetOffsetState
OFR4	OFFSET4	*LL_ADC_GetOffsetLevel*
		LL_ADC_SetOffset
	OFFSET4_CH	*LL_ADC_GetOffsetChannel*
		LL_ADC_SetOffset
	OFFSET4_EN	*LL_ADC_GetOffsetState*
		LL_ADC_SetOffset
		LL_ADC_SetOffsetState
SMPR1	SMP0	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP1	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP2	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP3	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP4	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP5	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP6	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP7	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP8	*LL_ADC_GetChannelSamplingTime*

Register	Field	Function
		LL_ADC_SetChannelSamplingTime
	SMP9	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMPPLUS	LL_ADC_GetSamplingTimeCommonConfig
		LL_ADC_SetSamplingTimeCommonConfig
SMPR2	SMP10	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP11	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP12	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP13	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP14	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP15	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP16	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP17	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP18	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
SQR1	L	LL_ADC_REG_GetSequencerLength
		LL_ADC_REG_SetSequencerLength
	SQ1	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ2	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ3	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ4	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
SQR2	SQ5	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ6	LL_ADC_REG_GetSequencerRanks

Register	Field	Function
		LL_ADC_REG_SetSequencerRanks
	SQ7	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ8	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ9	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
SQR3	SQ10	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ11	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ12	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ13	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ14	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
SQR4	SQ15	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ16	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
TR1	HT1	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
	LT1	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
TR2	HT2	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
	LT2	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
TR3	HT3	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds

Register	Field	Function
	LT3	*LL_ADC_ConfigAnalogWDThresholds*
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds

100.2 BUS

Table 27: Correspondence between BUS registers and BUS low-layer driver functions

Register	Field	Function
AHB1ENR	CRCEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DMA1EN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DMA2DEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DMA2EN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DMAMUX1EN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	FLASHEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	GFXMMUEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	TSCEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
AHB1RSTR	CRCRST	*LL_AHB1_GRP1_ForceReset*
		LL_AHB1_GRP1_ReleaseReset
	DMA1RST	*LL_AHB1_GRP1_ForceReset*
		LL_AHB1_GRP1_ReleaseReset
	DMA2DRST	*LL_AHB1_GRP1_ForceReset*

Register	Field	Function
		LL_AHB1_GRP1_ReleaseReset
	DMA2RST	*LL_AHB1_GRP1_ForceReset*
		LL_AHB1_GRP1_ReleaseReset
	DMAMUX1RST	*LL_AHB1_GRP1_ForceReset*
		LL_AHB1_GRP1_ReleaseReset
	FLASHRST	*LL_AHB1_GRP1_ForceReset*
		LL_AHB1_GRP1_ReleaseReset
	GFXMMURST	*LL_AHB1_GRP1_ForceReset*
		LL_AHB1_GRP1_ReleaseReset
	TSCRST	*LL_AHB1_GRP1_ForceReset*
		LL_AHB1_GRP1_ReleaseReset
AHB1SMENR	CRCSMEN	*LL_AHB1_GRP1_DisableClockStopSleep*
		LL_AHB1_GRP1_EnableClockStopSleep
	DMA1SMEN	*LL_AHB1_GRP1_DisableClockStopSleep*
		LL_AHB1_GRP1_EnableClockStopSleep
	DMA2DSMEN	*LL_AHB1_GRP1_DisableClockStopSleep*
		LL_AHB1_GRP1_EnableClockStopSleep
	DMA2SMEN	*LL_AHB1_GRP1_DisableClockStopSleep*
		LL_AHB1_GRP1_EnableClockStopSleep
	DMAMUX1SMEN	*LL_AHB1_GRP1_DisableClockStopSleep*
		LL_AHB1_GRP1_EnableClockStopSleep
	FLASHSMEN	*LL_AHB1_GRP1_DisableClockStopSleep*
		LL_AHB1_GRP1_EnableClockStopSleep
	GFXMMUSMEN	*LL_AHB1_GRP1_DisableClockStopSleep*
		LL_AHB1_GRP1_EnableClockStopSleep
	SRAM1SMEN	*LL_AHB1_GRP1_DisableClockStopSleep*
		LL_AHB1_GRP1_EnableClockStopSleep
	TSCSMEN	*LL_AHB1_GRP1_DisableClockStopSleep*
		LL_AHB1_GRP1_EnableClockStopSleep
AHB2ENR	ADCEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	AESEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	DCMIEN	*LL_AHB2_GRP1_DisableClock*

Register	Field	Function
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOAEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOBEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOCEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIODEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOEEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOFEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOGEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOHEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOIEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	HASHEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	OSPIMEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	OTGFSEN	*LL_AHB2_GRP1_DisableClock*

Register	Field	Function
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	RNGEN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	SDMMC1EN	*LL_AHB2_GRP1_DisableClock*
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
AHB2RSTR	ADCRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	AESRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	DCMIRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	GPIOARST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	GPIOBRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	GPIOCRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	GPIODRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	GPIOERST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	GPIOFRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	GPIOGRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	GPIOHRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	GPIOIRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	HASHRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	OSPIMRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset

Register	Field	Function
	OTGFSRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	RNGRST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
	SDMMC1RST	*LL_AHB2_GRP1_ForceReset*
		LL_AHB2_GRP1_ReleaseReset
AHB2SMENR	ADCSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	AESSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	DCMISMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	GPIOASMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	GPIOBSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	GPIOCSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	GPIODSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	GPIOESMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	GPIOFSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	GPIOGSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	GPIOHSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	GPIOISMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	HASHSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	OSPIMSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	OTGFSSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep

Register	Field	Function
	RNGSMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	SDMMC1SMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	SRAM2SMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
	SRAM3SMEN	*LL_AHB2_GRP1_DisableClockStopSleep*
		LL_AHB2_GRP1_EnableClockStopSleep
AHB3ENR	FMCEN	*LL_AHB3_GRP1_DisableClock*
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
	OSPI1EN	*LL_AHB3_GRP1_DisableClock*
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
	OSPI2EN	*LL_AHB3_GRP1_DisableClock*
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
	QSPIEN	*LL_AHB3_GRP1_DisableClock*
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
AHB3RSTR	FMCRST	*LL_AHB3_GRP1_ForceReset*
		LL_AHB3_GRP1_ReleaseReset
	OSPI1RST	*LL_AHB3_GRP1_ForceReset*
		LL_AHB3_GRP1_ReleaseReset
	OSPI2RST	*LL_AHB3_GRP1_ForceReset*
		LL_AHB3_GRP1_ReleaseReset
	QSPIRST	*LL_AHB3_GRP1_ForceReset*
		LL_AHB3_GRP1_ReleaseReset
AHB3SMENR	FMCSMEN	*LL_AHB3_GRP1_DisableClockStopSleep*
		LL_AHB3_GRP1_EnableClockStopSleep
	OSPI1SMEN	*LL_AHB3_GRP1_DisableClockStopSleep*
		LL_AHB3_GRP1_EnableClockStopSleep
	OSPI2SMEN	*LL_AHB3_GRP1_DisableClockStopSleep*
		LL_AHB3_GRP1_EnableClockStopSleep
	QSPISMEN	*LL_AHB3_GRP1_DisableClockStopSleep*
		LL_AHB3_GRP1_EnableClockStopSleep

Register	Field	Function
APB1ENR1	CAN1EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	CAN2EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	CRSEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	DAC1EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I2C1EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I2C2EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I2C3EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	LCDEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	LPTIM1EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	OPAMPEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	PWREN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	RTCAPBEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock

Register	Field	Function
	SPI2EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	SPI3EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM2EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM3EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM4EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM5EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM6EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM7EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	UART4EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	UART5EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART2EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART3EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock

Register	Field	Function
	USBFSEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	WWDGEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
APB1ENR2	I2C4EN	*LL_APB1_GRP2_DisableClock*
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	LPTIM2EN	*LL_APB1_GRP2_DisableClock*
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	LPUART1EN	*LL_APB1_GRP2_DisableClock*
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	SWPMI1EN	*LL_APB1_GRP2_DisableClock*
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
APB1RSTR1	CAN1RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	CAN2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	CRSRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	DAC1RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	I2C1RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	I2C2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	I2C3RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	LCDRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	LPTIM1RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset

Register	Field	Function
	OPAMPRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	PWRRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	SPI2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	SPI3RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM3RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM4RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM5RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM6RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM7RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	UART4RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	UART5RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	USART2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	USART3RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	USBFSRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
APB1RSTR2	I2C4RST	*LL_APB1_GRP2_ForceReset*
		LL_APB1_GRP2_ReleaseReset
	LPTIM2RST	*LL_APB1_GRP2_ForceReset*
		LL_APB1_GRP2_ReleaseReset
	LPUART1RST	*LL_APB1_GRP2_ForceReset*
		LL_APB1_GRP2_ReleaseReset

Register	Field	Function
	SWPMI1RST	*LL_APB1_GRP2_ForceReset*
		LL_APB1_GRP2_ReleaseReset
APB1SMENR1	CAN1SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	CAN2SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	CRSSMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	DAC1SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	I2C1SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	I2C2SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	I2C3SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	LCDSMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	LPTIM1SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	OPAMPSMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	PWRSMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	RTCAPBSMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	SPI2SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	SPI3SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	TIM2SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	TIM3SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep
	TIM4SMEN	*LL_APB1_GRP1_DisableClockStopSleep*
		LL_APB1_GRP1_EnableClockStopSleep

Register	Field	Function
	TIM5SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
	TIM6SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
	TIM7SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
	UART4SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
	UART5SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
	USART2SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
	USART3SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
	USBFSSMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
	WWDGSMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
APB1SMENR2	I2C4SMEN	LL_APB1_GRP2_DisableClockStopSleep
		LL_APB1_GRP2_EnableClockStopSleep
	LPTIM2SMEN	LL_APB1_GRP2_DisableClockStopSleep
		LL_APB1_GRP2_EnableClockStopSleep
	LPUART1SMEN	LL_APB1_GRP2_DisableClockStopSleep
		LL_APB1_GRP2_EnableClockStopSleep
	SWPMI1SMEN	LL_APB1_GRP2_DisableClockStopSleep
		LL_APB1_GRP2_EnableClockStopSleep
APB2ENR	DFSDM1EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	DSIEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	FWEN	LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	LTDCEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock

Register	Field	Function
		LL_APB2_GRP1_IsEnabledClock
	SAI1EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SAI2EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SDMMC1EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SPI1EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SYSCFGEN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM15EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM16EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM17EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM1EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM8EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	USART1EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
APB2RSTR	DFSDM1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset

Register	Field	Function
	DSIRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	LTDCRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	SAI1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	SAI2RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	SDMMC1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	SPI1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	SYSCFGRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM15RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM16RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM17RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM8RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	USART1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
APB2SMENR	DFSDM1SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	DSISMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	LTDCSMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	SAI1SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	SAI2SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep

Register	Field	Function
	SDMMC1SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	SPI1SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	SYSCFGSMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	TIM15SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	TIM16SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	TIM17SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	TIM1SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	TIM8SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep
	USART1SMEN	*LL_APB2_GRP1_DisableClockStopSleep*
		LL_APB2_GRP1_EnableClockStopSleep

100.3 COMP

Table 28: Correspondence between COMP registers and COMP low-layer driver functions

Register	Field	Function
CSR	BLANKING	*LL_COMP_GetOutputBlankingSource*
		LL_COMP_SetOutputBlankingSource
	BRGEN	*LL_COMP_ConfigInputs*
		LL_COMP_GetInputMinus
		LL_COMP_SetInputMinus
	EN	*LL_COMP_Disable*
		LL_COMP_Enable
		LL_COMP_IsEnabled
	HYST	*LL_COMP_GetInputHysteresis*
		LL_COMP_SetInputHysteresis
	INMSEL	*LL_COMP_ConfigInputs*
		LL_COMP_GetInputMinus
		LL_COMP_SetInputMinus
	INPSEL	*LL_COMP_ConfigInputs*

Register	Field	Function
		LL_COMP_GetInputPlus
		LL_COMP_SetInputPlus
	LOCK	*LL_COMP_IsLocked*
		LL_COMP_Lock
	POLARITY	*LL_COMP_GetOutputPolarity*
		LL_COMP_SetOutputPolarity
	PWRMODE	*LL_COMP_GetPowerMode*
		LL_COMP_SetPowerMode
	SCALEN	*LL_COMP_ConfigInputs*
		LL_COMP_GetInputMinus
		LL_COMP_SetInputMinus
	VALUE	*LL_COMP_ReadOutputLevel*
	WINMODE	*LL_COMP_GetCommonWindowMode*
		LL_COMP_SetCommonWindowMode

100.4 CORTEX

Table 29: Correspondence between CORTEX registers and CORTEX low-layer driver functions

Register	Field	Function
MPU_CTRL	ENABLE	*LL_MPU_Disable*
		LL_MPU_Enable
		LL_MPU_IsEnabled
MPU_RASR	AP	*LL_MPU_ConfigRegion*
	B	*LL_MPU_ConfigRegion*
	C	*LL_MPU_ConfigRegion*
	ENABLE	*LL_MPU_DisableRegion*
		LL_MPU_EnableRegion
	S	*LL_MPU_ConfigRegion*
	SIZE	*LL_MPU_ConfigRegion*
	XN	*LL_MPU_ConfigRegion*
MPU_RBAR	ADDR	*LL_MPU_ConfigRegion*
	REGION	*LL_MPU_ConfigRegion*
MPU_RNR	REGION	*LL_MPU_ConfigRegion*
		LL_MPU_DisableRegion
SCB_CPUID	ARCHITECTURE	*LL_CPUID_GetConstant*
	IMPLEMENTER	*LL_CPUID_GetImplementer*
	PARTNO	*LL_CPUID_GetParNo*

Register	Field	Function
	REVISION	*LL_CPUID_GetRevision*
	VARIANT	*LL_CPUID_GetVariant*
SCB_SCR	SEVEONPEND	*LL_LPM_DisableEventOnPend*
		LL_LPM_EnableEventOnPend
	SLEEPDEEP	*LL_LPM_EnableDeepSleep*
		LL_LPM_EnableSleep
	SLEEPONEXIT	*LL_LPM_DisableSleepOnExit*
		LL_LPM_EnableSleepOnExit
SCB_SHCSR	MEMFAULTENA	*LL_HANDLER_DisableFault*
		LL_HANDLER_EnableFault
STK_CTRL	CLKSOURCE	*LL_SYSTICK_GetClkSource*
		LL_SYSTICK_SetClkSource
	COUNTFLAG	*LL_SYSTICK_IsActiveCounterFlag*
	TICKINT	*LL_SYSTICK_DisableIT*
		LL_SYSTICK_EnableIT
		LL_SYSTICK_IsEnabledIT

100.5 CRC

Table 30: Correspondence between CRC registers and CRC low-layer driver functions

Register	Field	Function
CR	POLYSIZE	*LL_CRC_GetPolynomialSize*
		LL_CRC_SetPolynomialSize
	RESET	*LL_CRC_ResetCRCCalculationUnit*
	REV_IN	*LL_CRC_GetInputDataReverseMode*
		LL_CRC_SetInputDataReverseMode
	REV_OUT	*LL_CRC_GetOutputDataReverseMode*
		LL_CRC_SetOutputDataReverseMode
DR	DR	*LL_CRC_FeedData16*
		LL_CRC_FeedData32
		LL_CRC_FeedData8
		LL_CRC_ReadData16
		LL_CRC_ReadData32
		LL_CRC_ReadData7
		LL_CRC_ReadData8
IDR	IDR	*LL_CRC_Read_IDR*
		LL_CRC_Write_IDR

Register	Field	Function
INIT	INIT	*LL_CRC_GetInitialData*
		LL_CRC_SetInitialData
POL	POL	*LL_CRC_GetPolynomialCoef*
		LL_CRC_SetPolynomialCoef

100.6 CRS

Table 31: Correspondence between CRS registers and CRS low-layer driver functions

Register	Field	Function
CFGR	FELIM	*LL_CRS_ConfigSynchronization*
		LL_CRS_GetFreqErrorLimit
		LL_CRS_SetFreqErrorLimit
	RELOAD	*LL_CRS_ConfigSynchronization*
		LL_CRS_GetReloadCounter
		LL_CRS_SetReloadCounter
	SYNCDIV	*LL_CRS_ConfigSynchronization*
		LL_CRS_GetSyncDivider
		LL_CRS_SetSyncDivider
	SYNCPOL	*LL_CRS_ConfigSynchronization*
		LL_CRS_GetSyncPolarity
		LL_CRS_SetSyncPolarity
	SYNCSRC	*LL_CRS_ConfigSynchronization*
		LL_CRS_GetSyncSignalSource
		LL_CRS_SetSyncSignalSource
CR	AUTOTRIMEN	*LL_CRS_DisableAutoTrimming*
		LL_CRS_EnableAutoTrimming
		LL_CRS_IsEnabledAutoTrimming
	CEN	*LL_CRS_DisableFreqErrorCounter*
		LL_CRS_EnableFreqErrorCounter
		LL_CRS_IsEnabledFreqErrorCounter
	ERRIE	*LL_CRS_DisableIT_ERR*
		LL_CRS_EnableIT_ERR
		LL_CRS_IsEnabledIT_ERR
	ESYNCIE	*LL_CRS_DisableIT_ESYNC*
		LL_CRS_EnableIT_ESYNC
		LL_CRS_IsEnabledIT_ESYNC
	SWSYNC	*LL_CRS_GenerateEvent_SWSYNC*

Register	Field	Function
	SYNCOKIE	*LL_CRS_DisableIT_SYNCOK*
		LL_CRS_EnableIT_SYNCOK
		LL_CRS_IsEnabledIT_SYNCOK
	SYNCWARNIE	*LL_CRS_DisableIT_SYNCWARN*
		LL_CRS_EnableIT_SYNCWARN
		LL_CRS_IsEnabledIT_SYNCWARN
	TRIM	*LL_CRS_ConfigSynchronization*
		LL_CRS_GetHSI48SmoothTrimming
		LL_CRS_SetHSI48SmoothTrimming
ICR	ERRC	*LL_CRS_ClearFlag_ERR*
	ESYNCC	*LL_CRS_ClearFlag_ESYNC*
	SYNCOKC	*LL_CRS_ClearFlag_SYNCOK*
	SYNCWARNC	*LL_CRS_ClearFlag_SYNCWARN*
ISR	ERRF	*LL_CRS_IsActiveFlag_ERR*
	ESYNCF	*LL_CRS_IsActiveFlag_ESYNC*
	FECAP	*LL_CRS_GetFreqErrorCapture*
	FEDIR	*LL_CRS_GetFreqErrorDirection*
	SYNCERR	*LL_CRS_IsActiveFlag_SYNCERR*
	SYNCMISS	*LL_CRS_IsActiveFlag_SYNCMISS*
	SYNCOKF	*LL_CRS_IsActiveFlag_SYNCOK*
	SYNCWARNF	*LL_CRS_IsActiveFlag_SYNCWARN*
	TRIMOVF	*LL_CRS_IsActiveFlag_TRIMOVF*

100.7 DAC

Table 32: Correspondence between DAC registers and DAC low-layer driver functions

Register	Field	Function
CCR	OTRIM1	*LL_DAC_GetTrimmingValue*
		LL_DAC_SetTrimmingValue
	OTRIM2	*LL_DAC_GetTrimmingValue*
		LL_DAC_SetTrimmingValue
CR	CEN1	*LL_DAC_GetMode*
		LL_DAC_SetMode
	CEN2	*LL_DAC_GetMode*
		LL_DAC_SetMode
	DMAEN1	*LL_DAC_DisableDMAReq*
		LL_DAC_EnableDMAReq

Register	Field	Function
		LL_DAC_IsDMAReqEnabled
	DMAEN2	LL_DAC_DisableDMAReq
		LL_DAC_EnableDMAReq
		LL_DAC_IsDMAReqEnabled
	DMAUDRIE1	LL_DAC_DisableIT_DMAUDR1
		LL_DAC_EnableIT_DMAUDR1
		LL_DAC_IsEnabledIT_DMAUDR1
	DMAUDRIE2	LL_DAC_DisableIT_DMAUDR2
		LL_DAC_EnableIT_DMAUDR2
		LL_DAC_IsEnabledIT_DMAUDR2
	EN1	LL_DAC_Disable
		LL_DAC_Enable
		LL_DAC_IsEnabled
	EN2	LL_DAC_Disable
		LL_DAC_Enable
		LL_DAC_IsEnabled
	MAMP1	LL_DAC_GetWaveNoiseLFSR
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	MAMP2	LL_DAC_GetWaveNoiseLFSR
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	MODE1	LL_DAC_ConfigOutput
		LL_DAC_GetOutputBuffer
		LL_DAC_GetOutputConnection
		LL_DAC_GetOutputMode
		LL_DAC_SetOutputBuffer
		LL_DAC_SetOutputConnection
		LL_DAC_SetOutputMode
	MODE2	LL_DAC_ConfigOutput
		LL_DAC_GetOutputBuffer
		LL_DAC_GetOutputConnection
		LL_DAC_GetOutputMode
		LL_DAC_SetOutputBuffer

Register	Field	Function
		LL_DAC_SetOutputConnection
		LL_DAC_SetOutputMode
	TEN1	*LL_DAC_DisableTrigger*
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TEN2	*LL_DAC_DisableTrigger*
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TSEL1	*LL_DAC_GetTriggerSource*
		LL_DAC_SetTriggerSource
	TSEL2	*LL_DAC_GetTriggerSource*
		LL_DAC_SetTriggerSource
	WAVE1	*LL_DAC_GetWaveAutoGeneration*
		LL_DAC_SetWaveAutoGeneration
	WAVE2	*LL_DAC_GetWaveAutoGeneration*
		LL_DAC_SetWaveAutoGeneration
DHR12L1	DACC1DHR	*LL_DAC_ConvertData12LeftAligned*
		LL_DAC_DMA_GetRegAddr
DHR12L2	DACC2DHR	*LL_DAC_ConvertData12LeftAligned*
		LL_DAC_DMA_GetRegAddr
DHR12LD	DACC1DHR	*LL_DAC_ConvertDualData12LeftAligned*
	DACC2DHR	*LL_DAC_ConvertDualData12LeftAligned*
DHR12R1	DACC1DHR	*LL_DAC_ConvertData12RightAligned*
		LL_DAC_DMA_GetRegAddr
DHR12R2	DACC2DHR	*LL_DAC_ConvertData12RightAligned*
		LL_DAC_DMA_GetRegAddr
DHR12RD	DACC1DHR	*LL_DAC_ConvertDualData12RightAligned*
	DACC2DHR	*LL_DAC_ConvertDualData12RightAligned*
DHR8R1	DACC1DHR	*LL_DAC_ConvertData8RightAligned*
		LL_DAC_DMA_GetRegAddr
DHR8R2	DACC2DHR	*LL_DAC_ConvertData8RightAligned*
		LL_DAC_DMA_GetRegAddr
DHR8RD	DACC1DHR	*LL_DAC_ConvertDualData8RightAligned*
	DACC2DHR	*LL_DAC_ConvertDualData8RightAligned*
DOR1	DACC1DOR	*LL_DAC_RetrieveOutputData*
DOR2	DACC2DOR	*LL_DAC_RetrieveOutputData*

Register	Field	Function
SHHR	THOLD1	*LL_DAC_GetSampleAndHoldHoldTime*
		LL_DAC_SetSampleAndHoldHoldTime
	THOLD2	*LL_DAC_GetSampleAndHoldHoldTime*
		LL_DAC_SetSampleAndHoldHoldTime
SHRR	TREFRESH1	*LL_DAC_GetSampleAndHoldRefreshTime*
		LL_DAC_SetSampleAndHoldRefreshTime
	TREFRESH2	*LL_DAC_GetSampleAndHoldRefreshTime*
		LL_DAC_SetSampleAndHoldRefreshTime
SHSR1	TSAMPLE1	*LL_DAC_GetSampleAndHoldSampleTime*
		LL_DAC_SetSampleAndHoldSampleTime
SHSR2	TSAMPLE2	*LL_DAC_GetSampleAndHoldSampleTime*
		LL_DAC_SetSampleAndHoldSampleTime
SR	BWST1	*LL_DAC_IsActiveFlag_BWST1*
	BWST2	*LL_DAC_IsActiveFlag_BWST2*
	CAL_FLAG1	*LL_DAC_IsActiveFlag_CAL1*
	CAL_FLAG2	*LL_DAC_IsActiveFlag_CAL2*
	DMAUDR1	*LL_DAC_ClearFlag_DMAUDR1*
		LL_DAC_IsActiveFlag_DMAUDR1
	DMAUDR2	*LL_DAC_ClearFlag_DMAUDR2*
		LL_DAC_IsActiveFlag_DMAUDR2
SWTRIGR	SWTRIG1	*LL_DAC_TrigSWConversion*
	SWTRIG2	*LL_DAC_TrigSWConversion*

100.8 DMA

Table 33: Correspondence between DMA registers and DMA low-layer driver functions

Register	Field	Function
CCR	CIRC	*LL_DMA_ConfigTransfer*
		LL_DMA_GetMode
		LL_DMA_SetMode
	DIR	*LL_DMA_ConfigTransfer*
		LL_DMA_GetDataTransferDirection
		LL_DMA_SetDataTransferDirection
	EN	*LL_DMA_DisableChannel*
		LL_DMA_EnableChannel
		LL_DMA_IsEnabledChannel
	HTIE	*LL_DMA_DisableIT_HT*

Register	Field	Function
		LL_DMA_EnableIT_HT
		LL_DMA_IsEnabledIT_HT
	MEM2MEM	LL_DMA_ConfigTransfer
		LL_DMA_GetDataTransferDirection
		LL_DMA_SetDataTransferDirection
	MINC	LL_DMA_ConfigTransfer
		LL_DMA_GetMemoryIncMode
		LL_DMA_SetMemoryIncMode
	MSIZE	LL_DMA_ConfigTransfer
		LL_DMA_GetMemorySize
		LL_DMA_SetMemorySize
	PINC	LL_DMA_ConfigTransfer
		LL_DMA_GetPeriphIncMode
		LL_DMA_SetPeriphIncMode
	PL	LL_DMA_ConfigTransfer
		LL_DMA_GetChannelPriorityLevel
		LL_DMA_SetChannelPriorityLevel
	PSIZE	LL_DMA_ConfigTransfer
		LL_DMA_GetPeriphSize
		LL_DMA_SetPeriphSize
	TCIE	LL_DMA_DisableIT_TC
		LL_DMA_EnableIT_TC
		LL_DMA_IsEnabledIT_TC
	TEIE	LL_DMA_DisableIT_TE
		LL_DMA_EnableIT_TE
		LL_DMA_IsEnabledIT_TE
CMAR	MA	LL_DMA_ConfigAddresses
		LL_DMA_GetM2MDstAddress
		LL_DMA_GetMemoryAddress
		LL_DMA_SetM2MDstAddress
		LL_DMA_SetMemoryAddress
CNDTR	NDT	LL_DMA_GetDataLength
		LL_DMA_SetDataLength
CPAR	PA	LL_DMA_ConfigAddresses
		LL_DMA_GetM2MSrcAddress
		LL_DMA_GetPeriphAddress

Register	Field	Function
		LL_DMA_SetM2MSrcAddress
		LL_DMA_SetPeriphAddress
CxCR	DMAREQ_ID	LL_DMA_GetPeriphRequest
		LL_DMA_SetPeriphRequest
IFCR	CGIF1	LL_DMA_ClearFlag_GI1
	CGIF2	LL_DMA_ClearFlag_GI2
	CGIF3	LL_DMA_ClearFlag_GI3
	CGIF4	LL_DMA_ClearFlag_GI4
	CGIF5	LL_DMA_ClearFlag_GI5
	CGIF6	LL_DMA_ClearFlag_GI6
	CGIF7	LL_DMA_ClearFlag_GI7
	CHTIF1	LL_DMA_ClearFlag_HT1
	CHTIF2	LL_DMA_ClearFlag_HT2
	CHTIF3	LL_DMA_ClearFlag_HT3
	CHTIF4	LL_DMA_ClearFlag_HT4
	CHTIF5	LL_DMA_ClearFlag_HT5
	CHTIF6	LL_DMA_ClearFlag_HT6
	CHTIF7	LL_DMA_ClearFlag_HT7
	CTCIF1	LL_DMA_ClearFlag_TC1
	CTCIF2	LL_DMA_ClearFlag_TC2
	CTCIF3	LL_DMA_ClearFlag_TC3
	CTCIF4	LL_DMA_ClearFlag_TC4
	CTCIF5	LL_DMA_ClearFlag_TC5
	CTCIF6	LL_DMA_ClearFlag_TC6
	CTCIF7	LL_DMA_ClearFlag_TC7
	CTEIF1	LL_DMA_ClearFlag_TE1
	CTEIF2	LL_DMA_ClearFlag_TE2
	CTEIF3	LL_DMA_ClearFlag_TE3
	CTEIF4	LL_DMA_ClearFlag_TE4
	CTEIF5	LL_DMA_ClearFlag_TE5
	CTEIF6	LL_DMA_ClearFlag_TE6
	CTEIF7	LL_DMA_ClearFlag_TE7
ISR	GIF1	LL_DMA_IsActiveFlag_GI1
	GIF2	LL_DMA_IsActiveFlag_GI2
	GIF3	LL_DMA_IsActiveFlag_GI3
	GIF4	LL_DMA_IsActiveFlag_GI4

Register	Field	Function
	GIF5	*LL_DMA_IsActiveFlag_GI5*
	GIF6	*LL_DMA_IsActiveFlag_GI6*
	GIF7	*LL_DMA_IsActiveFlag_GI7*
	HTIF1	*LL_DMA_IsActiveFlag_HT1*
	HTIF2	*LL_DMA_IsActiveFlag_HT2*
	HTIF3	*LL_DMA_IsActiveFlag_HT3*
	HTIF4	*LL_DMA_IsActiveFlag_HT4*
	HTIF5	*LL_DMA_IsActiveFlag_HT5*
	HTIF6	*LL_DMA_IsActiveFlag_HT6*
	HTIF7	*LL_DMA_IsActiveFlag_HT7*
	TCIF1	*LL_DMA_IsActiveFlag_TC1*
	TCIF2	*LL_DMA_IsActiveFlag_TC2*
	TCIF3	*LL_DMA_IsActiveFlag_TC3*
	TCIF4	*LL_DMA_IsActiveFlag_TC4*
	TCIF5	*LL_DMA_IsActiveFlag_TC5*
	TCIF6	*LL_DMA_IsActiveFlag_TC6*
	TCIF7	*LL_DMA_IsActiveFlag_TC7*
	TEIF1	*LL_DMA_IsActiveFlag_TE1*
	TEIF2	*LL_DMA_IsActiveFlag_TE2*
	TEIF3	*LL_DMA_IsActiveFlag_TE3*
	TEIF4	*LL_DMA_IsActiveFlag_TE4*
	TEIF5	*LL_DMA_IsActiveFlag_TE5*
	TEIF6	*LL_DMA_IsActiveFlag_TE6*
	TEIF7	*LL_DMA_IsActiveFlag_TE7*

100.9 DMA2D

Table 34: Correspondence between DMA2D registers and DMA2D low-layer driver functions

Register	Field	Function
AMTCR	DT	*LL_DMA2D_GetDeadTime*
		LL_DMA2D_SetDeadTime
	EN	*LL_DMA2D_DisableDeadTime*
		LL_DMA2D_EnableDeadTime
		LL_DMA2D_IsEnabledDeadTime
BGCMAR	MA	*LL_DMA2D_BGND_GetCLUTMemAddr*
		LL_DMA2D_BGND_SetCLUTMemAddr
BGCOLR	BLUE	*LL_DMA2D_BGND_GetBlueColor*

Register	Field	Function
		LL_DMA2D_BGND_SetBlueColor
		LL_DMA2D_BGND_SetColor
	GREEN	LL_DMA2D_BGND_GetGreenColor
		LL_DMA2D_BGND_SetColor
		LL_DMA2D_BGND_SetGreenColor
	RED	LL_DMA2D_BGND_GetRedColor
		LL_DMA2D_BGND_SetColor
		LL_DMA2D_BGND_SetRedColor
BGMAR	MA	LL_DMA2D_BGND_GetMemAddr
		LL_DMA2D_BGND_SetMemAddr
BGOR	LO	LL_DMA2D_BGND_GetLineOffset
		LL_DMA2D_BGND_SetLineOffset
BGPFCCR	AI	LL_DMA2D_BGND_GetAlphaInvMode
		LL_DMA2D_BGND_SetAlphaInvMode
	ALPHA	LL_DMA2D_BGND_GetAlpha
		LL_DMA2D_BGND_SetAlpha
	AM	LL_DMA2D_BGND_GetAlphaMode
		LL_DMA2D_BGND_SetAlphaMode
	CCM	LL_DMA2D_BGND_GetCLUTColorMode
		LL_DMA2D_BGND_SetCLUTColorMode
	CM	LL_DMA2D_BGND_GetColorMode
		LL_DMA2D_BGND_SetColorMode
	CS	LL_DMA2D_BGND_GetCLUTSize
		LL_DMA2D_BGND_SetCLUTSize
	RBS	LL_DMA2D_BGND_GetRBSwapMode
		LL_DMA2D_BGND_SetRBSwapMode
	START	LL_DMA2D_BGND_EnableCLUTLoad
		LL_DMA2D_BGND_IsEnabledCLUTLoad
CR	ABORT	LL_DMA2D_Abort
		LL_DMA2D_IsAborted
	CAEIE	LL_DMA2D_DisableIT_CAE
		LL_DMA2D_EnableIT_CAE
		LL_DMA2D_IsEnabledIT_CAE
	CEIE	LL_DMA2D_DisableIT_CE
		LL_DMA2D_EnableIT_CE
		LL_DMA2D_IsEnabledIT_CE

Register	Field	Function
	CTCIE	LL_DMA2D_DisableIT_CTC
		LL_DMA2D_EnableIT_CTC
		LL_DMA2D_IsEnabledIT_CTC
	LOM	LL_DMA2D_GetLineOffsetMode
		LL_DMA2D_SetLineOffsetMode
	MODE	LL_DMA2D_GetMode
		LL_DMA2D_SetMode
	START	LL_DMA2D_IsTransferOngoing
		LL_DMA2D_Start
	SUSP	LL_DMA2D_IsSuspended
		LL_DMA2D_Resume
		LL_DMA2D_Suspend
	TCIE	LL_DMA2D_DisableIT_TC
		LL_DMA2D_EnableIT_TC
		LL_DMA2D_IsEnabledIT_TC
	TEIE	LL_DMA2D_DisableIT_TE
		LL_DMA2D_EnableIT_TE
		LL_DMA2D_IsEnabledIT_TE
	TWIE	LL_DMA2D_DisableIT_TW
		LL_DMA2D_EnableIT_TW
		LL_DMA2D_IsEnabledIT_TW
FGCMAR	MA	LL_DMA2D_FGND_GetCLUTMemAddr
		LL_DMA2D_FGND_SetCLUTMemAddr
FGCOLR	BLUE	LL_DMA2D_FGND_GetBlueColor
		LL_DMA2D_FGND_SetBlueColor
		LL_DMA2D_FGND_SetColor
	GREEN	LL_DMA2D_FGND_GetGreenColor
		LL_DMA2D_FGND_SetColor
		LL_DMA2D_FGND_SetGreenColor
	RED	LL_DMA2D_FGND_GetRedColor
		LL_DMA2D_FGND_SetColor
		LL_DMA2D_FGND_SetRedColor
FGMAR	MA	LL_DMA2D_FGND_GetMemAddr
		LL_DMA2D_FGND_SetMemAddr
FGOR	LO	LL_DMA2D_FGND_GetLineOffset
		LL_DMA2D_FGND_SetLineOffset

Register	Field	Function
FGPFCCR	AI	LL_DMA2D_FGND_GetAlphaInvMode
		LL_DMA2D_FGND_SetAlphaInvMode
	ALPHA	LL_DMA2D_FGND_GetAlpha
		LL_DMA2D_FGND_SetAlpha
	AM	LL_DMA2D_FGND_GetAlphaMode
		LL_DMA2D_FGND_SetAlphaMode
	CCM	LL_DMA2D_FGND_GetCLUTColorMode
		LL_DMA2D_FGND_SetCLUTColorMode
	CM	LL_DMA2D_FGND_GetColorMode
		LL_DMA2D_FGND_SetColorMode
	CS	LL_DMA2D_FGND_GetCLUTSize
		LL_DMA2D_FGND_SetCLUTSize
	RBS	LL_DMA2D_FGND_GetRBSwapMode
		LL_DMA2D_FGND_SetRBSwapMode
	START	LL_DMA2D_FGND_EnableCLUTLoad
		LL_DMA2D_FGND_IsEnabledCLUTLoad
IFCR	CAECIF	LL_DMA2D_ClearFlag_CAE
	CCEIF	LL_DMA2D_ClearFlag_CE
	CCTCIF	LL_DMA2D_ClearFlag_CTC
	CTCIF	LL_DMA2D_ClearFlag_TC
	CTEIF	LL_DMA2D_ClearFlag_TE
	CTWIF	LL_DMA2D_ClearFlag_TW
ISR	CAEIF	LL_DMA2D_IsActiveFlag_CAE
	CEIF	LL_DMA2D_IsActiveFlag_CE
	CTCIF	LL_DMA2D_IsActiveFlag_CTC
	TCIF	LL_DMA2D_IsActiveFlag_TC
	TEIF	LL_DMA2D_IsActiveFlag_TE
	TWIF	LL_DMA2D_IsActiveFlag_TW
LWR	LW	LL_DMA2D_GetLineWatermark
		LL_DMA2D_SetLineWatermark
NLR	NL	LL_DMA2D_GetNbrOfLines
		LL_DMA2D_SetNbrOfLines
	PL	LL_DMA2D_GetNbrOfPixelsPerLines
		LL_DMA2D_SetNbrOfPixelsPerLines
OCOLR	ALPHA	LL_DMA2D_GetOutputColor
		LL_DMA2D_SetOutputColor

Register	Field	Function
	BLUE	*LL_DMA2D_GetOutputColor*
		LL_DMA2D_SetOutputColor
	GREEN	*LL_DMA2D_GetOutputColor*
		LL_DMA2D_SetOutputColor
	RED	*LL_DMA2D_GetOutputColor*
		LL_DMA2D_SetOutputColor
OMAR	MA	*LL_DMA2D_GetOutputMemAddr*
		LL_DMA2D_SetOutputMemAddr
OOR	LO	*LL_DMA2D_GetLineOffset*
		LL_DMA2D_SetLineOffset
OPFCCR	AI	*LL_DMA2D_GetOutputAlphaInvMode*
		LL_DMA2D_SetOutputAlphaInvMode
	CM	*LL_DMA2D_GetOutputColorMode*
		LL_DMA2D_SetOutputColorMode
	RBS	*LL_DMA2D_GetOutputRBSwapMode*
		LL_DMA2D_SetOutputRBSwapMode
	SB	*LL_DMA2D_GetOutputSwapMode*
		LL_DMA2D_SetOutputSwapMode

100.10 DMAMUX

Table 35: Correspondence between DMAMUX registers and DMAMUX low-layer driver functions

Register	Field	Function
CFR	CSOF0	*LL_DMAMUX_ClearFlag_SO0*
	CSOF1	*LL_DMAMUX_ClearFlag_SO1*
	CSOF10	*LL_DMAMUX_ClearFlag_SO10*
	CSOF11	*LL_DMAMUX_ClearFlag_SO11*
	CSOF12	*LL_DMAMUX_ClearFlag_SO12*
	CSOF13	*LL_DMAMUX_ClearFlag_SO13*
	CSOF2	*LL_DMAMUX_ClearFlag_SO2*
	CSOF3	*LL_DMAMUX_ClearFlag_SO3*
	CSOF4	*LL_DMAMUX_ClearFlag_SO4*
	CSOF5	*LL_DMAMUX_ClearFlag_SO5*
	CSOF6	*LL_DMAMUX_ClearFlag_SO6*
	CSOF7	*LL_DMAMUX_ClearFlag_SO7*
	CSOF8	*LL_DMAMUX_ClearFlag_SO8*

Register	Field	Function
	CSOF9	*LL_DMAMUX_ClearFlag_SO9*
CSR	SOF0	*LL_DMAMUX_IsActiveFlag_SO0*
	SOF1	*LL_DMAMUX_IsActiveFlag_SO1*
	SOF10	*LL_DMAMUX_IsActiveFlag_SO10*
	SOF11	*LL_DMAMUX_IsActiveFlag_SO11*
	SOF12	*LL_DMAMUX_IsActiveFlag_SO12*
	SOF13	*LL_DMAMUX_IsActiveFlag_SO13*
	SOF2	*LL_DMAMUX_IsActiveFlag_SO2*
	SOF3	*LL_DMAMUX_IsActiveFlag_SO3*
	SOF4	*LL_DMAMUX_IsActiveFlag_SO4*
	SOF5	*LL_DMAMUX_IsActiveFlag_SO5*
	SOF6	*LL_DMAMUX_IsActiveFlag_SO6*
	SOF7	*LL_DMAMUX_IsActiveFlag_SO7*
	SOF8	*LL_DMAMUX_IsActiveFlag_SO8*
	SOF9	*LL_DMAMUX_IsActiveFlag_SO9*
CxCR	DMAREQ_ID	*LL_DMAMUX_GetRequestID*
		LL_DMAMUX_SetRequestID
	EGE	*LL_DMAMUX_DisableEventGeneration*
		LL_DMAMUX_EnableEventGeneration
		LL_DMAMUX_IsEnabledEventGeneration
	NBREQ	*LL_DMAMUX_GetSyncRequestNb*
		LL_DMAMUX_SetSyncRequestNb
	SE	*LL_DMAMUX_DisableSync*
		LL_DMAMUX_EnableSync
		LL_DMAMUX_IsEnabledSync
	SOIE	*LL_DMAMUX_DisableIT_SO*
		LL_DMAMUX_EnableIT_SO
		LL_DMAMUX_IsEnabledIT_SO
	SPOL	*LL_DMAMUX_GetSyncPolarity*
		LL_DMAMUX_SetSyncPolarity
	SYNC_ID	*LL_DMAMUX_GetSyncID*
		LL_DMAMUX_SetSyncID
RGCFR	COF0	*LL_DMAMUX_ClearFlag_RGO0*
	COF1	*LL_DMAMUX_ClearFlag_RGO1*
	COF2	*LL_DMAMUX_ClearFlag_RGO2*
	COF3	*LL_DMAMUX_ClearFlag_RGO3*

Register	Field	Function
RGSR	OF0	*LL_DMAMUX_IsActiveFlag_RGO0*
	OF1	*LL_DMAMUX_IsActiveFlag_RGO1*
	OF2	*LL_DMAMUX_IsActiveFlag_RGO2*
	OF3	*LL_DMAMUX_IsActiveFlag_RGO3*
RGxCR	GE	*LL_DMAMUX_DisableRequestGen*
		LL_DMAMUX_EnableRequestGen
		LL_DMAMUX_IsEnabledRequestGen
	GNBREQ	*LL_DMAMUX_GetGenRequestNb*
		LL_DMAMUX_SetGenRequestNb
	GPOL	*LL_DMAMUX_GetRequestGenPolarity*
		LL_DMAMUX_SetRequestGenPolarity
	OIE	*LL_DMAMUX_DisableIT_RGO*
		LL_DMAMUX_EnableIT_RGO
		LL_DMAMUX_IsEnabledIT_RGO
	SIG_ID	*LL_DMAMUX_GetRequestSignalID*
		LL_DMAMUX_SetRequestSignalID

100.11 EXTI

Table 36: Correspondence between EXTI registers and EXTI low-layer driver functions

Register	Field	Function
EMR1	EMx	*LL_EXTI_DisableEvent_0_31*
		LL_EXTI_EnableEvent_0_31
		LL_EXTI_IsEnabledEvent_0_31
EMR2	EMx	*LL_EXTI_DisableEvent_32_63*
		LL_EXTI_EnableEvent_32_63
		LL_EXTI_IsEnabledEvent_32_63
FTSR1	FTx	*LL_EXTI_DisableFallingTrig_0_31*
		LL_EXTI_EnableFallingTrig_0_31
		LL_EXTI_IsEnabledFallingTrig_0_31
FTSR2	FTx	*LL_EXTI_DisableFallingTrig_32_63*
		LL_EXTI_EnableFallingTrig_32_63
		LL_EXTI_IsEnabledFallingTrig_32_63
IMR1	IMx	*LL_EXTI_DisableIT_0_31*
		LL_EXTI_EnableIT_0_31
		LL_EXTI_IsEnabledIT_0_31
IMR2	IMx	*LL_EXTI_DisableIT_32_63*

Register	Field	Function
		LL_EXTI_EnableIT_32_63
		LL_EXTI_IsEnabledIT_32_63
PR1	PIFx	*LL_EXTI_ClearFlag_0_31*
		LL_EXTI_IsActiveFlag_0_31
		LL_EXTI_ReadFlag_0_31
PR2	PIFx	*LL_EXTI_ClearFlag_32_63*
		LL_EXTI_IsActiveFlag_32_63
		LL_EXTI_ReadFlag_32_63
RTSR1	RTx	*LL_EXTI_DisableRisingTrig_0_31*
		LL_EXTI_EnableRisingTrig_0_31
		LL_EXTI_IsEnabledRisingTrig_0_31
RTSR2	RTx	*LL_EXTI_DisableRisingTrig_32_63*
		LL_EXTI_EnableRisingTrig_32_63
		LL_EXTI_IsEnabledRisingTrig_32_63
SWIER1	SWIx	*LL_EXTI_GenerateSWI_0_31*
SWIER2	SWIx	*LL_EXTI_GenerateSWI_32_63*

100.12 GPIO

Table 37: Correspondence between GPIO registers and GPIO low-layer driver functions

Register	Field	Function
AFRH	AFSELy	*LL_GPIO_GetAFPin_8_15*
		LL_GPIO_SetAFPin_8_15
AFRL	AFSELy	*LL_GPIO_GetAFPin_0_7*
		LL_GPIO_SetAFPin_0_7
BRR	BRy	*LL_GPIO_ResetOutputPin*
BSRR	BSy	*LL_GPIO_SetOutputPin*
IDR	IDy	*LL_GPIO_IsInputPinSet*
		LL_GPIO_ReadInputPort
LCKR	LCKK	*LL_GPIO_IsAnyPinLocked*
		LL_GPIO_LockPin
	LCKy	*LL_GPIO_IsPinLocked*
MODER	MODEy	*LL_GPIO_GetPinMode*
		LL_GPIO_SetPinMode
ODR	ODy	*LL_GPIO_IsOutputPinSet*
		LL_GPIO_ReadOutputPort
		LL_GPIO_TogglePin

Register	Field	Function
		LL_GPIO_WriteOutputPort
OSPEEDR	OSPEEDy	*LL_GPIO_GetPinSpeed*
		LL_GPIO_SetPinSpeed
OTYPER	OTy	*LL_GPIO_GetPinOutputType*
		LL_GPIO_SetPinOutputType
PUPDR	PUPDy	*LL_GPIO_GetPinPull*
		LL_GPIO_SetPinPull

100.13 I2C

Table 38: Correspondence between I2C registers and I2C low-layer driver functions

Register	Field	Function
CR1	ADDRIE	*LL_I2C_DisableIT_ADDR*
		LL_I2C_EnableIT_ADDR
		LL_I2C_IsEnabledIT_ADDR
	ALERTEN	*LL_I2C_DisableSMBusAlert*
		LL_I2C_EnableSMBusAlert
		LL_I2C_IsEnabledSMBusAlert
	ANFOFF	*LL_I2C_ConfigFilters*
		LL_I2C_DisableAnalogFilter
		LL_I2C_EnableAnalogFilter
		LL_I2C_IsEnabledAnalogFilter
	DNF	*LL_I2C_ConfigFilters*
		LL_I2C_GetDigitalFilter
		LL_I2C_SetDigitalFilter
	ERRIE	*LL_I2C_DisableIT_ERR*
		LL_I2C_EnableIT_ERR
		LL_I2C_IsEnabledIT_ERR
	GCEN	*LL_I2C_DisableGeneralCall*
		LL_I2C_EnableGeneralCall
		LL_I2C_IsEnabledGeneralCall
	NACKIE	*LL_I2C_DisableIT_NACK*
		LL_I2C_EnableIT_NACK
		LL_I2C_IsEnabledIT_NACK
	NOSTRETCH	*LL_I2C_DisableClockStretching*
		LL_I2C_EnableClockStretching
		LL_I2C_IsEnabledClockStretching

Register	Field	Function
	PE	*LL_I2C_Disable*
		LL_I2C_Enable
		LL_I2C_IsEnabled
	PECEN	*LL_I2C_DisableSMBusPEC*
		LL_I2C_EnableSMBusPEC
		LL_I2C_IsEnabledSMBusPEC
	RXDMAEN	*LL_I2C_DisableDMAReq_RX*
		LL_I2C_EnableDMAReq_RX
		LL_I2C_IsEnabledDMAReq_RX
	RXIE	*LL_I2C_DisableIT_RX*
		LL_I2C_EnableIT_RX
		LL_I2C_IsEnabledIT_RX
	SBC	*LL_I2C_DisableSlaveByteControl*
		LL_I2C_EnableSlaveByteControl
		LL_I2C_IsEnabledSlaveByteControl
	SMBDEN	*LL_I2C_GetMode*
		LL_I2C_SetMode
	SMBHEN	*LL_I2C_GetMode*
		LL_I2C_SetMode
	STOPIE	*LL_I2C_DisableIT_STOP*
		LL_I2C_EnableIT_STOP
		LL_I2C_IsEnabledIT_STOP
	TCIE	*LL_I2C_DisableIT_TC*
		LL_I2C_EnableIT_TC
		LL_I2C_IsEnabledIT_TC
	TXDMAEN	*LL_I2C_DisableDMAReq_TX*
		LL_I2C_EnableDMAReq_TX
		LL_I2C_IsEnabledDMAReq_TX
	TXIE	*LL_I2C_DisableIT_TX*
		LL_I2C_EnableIT_TX
		LL_I2C_IsEnabledIT_TX
	WUPEN	*LL_I2C_DisableWakeUpFromStop*
		LL_I2C_EnableWakeUpFromStop
		LL_I2C_IsEnabledWakeUpFromStop
CR2	ADD10	*LL_I2C_GetMasterAddressingMode*
		LL_I2C_HandleTransfer

Register	Field	Function
		LL_I2C_SetMasterAddressingMode
	AUTOEND	*LL_I2C_DisableAutoEndMode*
		LL_I2C_EnableAutoEndMode
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAutoEndMode
	HEAD10R	*LL_I2C_DisableAuto10BitRead*
		LL_I2C_EnableAuto10BitRead
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAuto10BitRead
	NACK	*LL_I2C_AcknowledgeNextData*
	NBYTES	*LL_I2C_GetTransferSize*
		LL_I2C_HandleTransfer
		LL_I2C_SetTransferSize
	PECBYTE	*LL_I2C_EnableSMBusPECCompare*
		LL_I2C_IsEnabledSMBusPECCompare
	RD_WRN	*LL_I2C_GetTransferRequest*
		LL_I2C_HandleTransfer
		LL_I2C_SetTransferRequest
	RELOAD	*LL_I2C_DisableReloadMode*
		LL_I2C_EnableReloadMode
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledReloadMode
	SADD	*LL_I2C_GetSlaveAddr*
		LL_I2C_HandleTransfer
		LL_I2C_SetSlaveAddr
	START	*LL_I2C_GenerateStartCondition*
		LL_I2C_HandleTransfer
	STOP	*LL_I2C_GenerateStopCondition*
		LL_I2C_HandleTransfer
ICR	ADDRCF	*LL_I2C_ClearFlag_ADDR*
	ALERTCF	*LL_I2C_ClearSMBusFlag_ALERT*
	ARLOCF	*LL_I2C_ClearFlag_ARLO*
	BERRCF	*LL_I2C_ClearFlag_BERR*
	NACKCF	*LL_I2C_ClearFlag_NACK*
	OVRCF	*LL_I2C_ClearFlag_OVR*
	PECCF	*LL_I2C_ClearSMBusFlag_PECERR*

Register	Field	Function
	STOPCF	*LL_I2C_ClearFlag_STOP*
	TIMOUTCF	*LL_I2C_ClearSMBusFlag_TIMEOUT*
ISR	ADDCODE	*LL_I2C_GetAddressMatchCode*
	ADDR	*LL_I2C_IsActiveFlag_ADDR*
	ALERT	*LL_I2C_IsActiveSMBusFlag_ALERT*
	ARLO	*LL_I2C_IsActiveFlag_ARLO*
	BERR	*LL_I2C_IsActiveFlag_BERR*
	BUSY	*LL_I2C_IsActiveFlag_BUSY*
	DIR	*LL_I2C_GetTransferDirection*
	NACKF	*LL_I2C_IsActiveFlag_NACK*
	OVR	*LL_I2C_IsActiveFlag_OVR*
	PECERR	*LL_I2C_IsActiveSMBusFlag_PECERR*
	RXNE	*LL_I2C_IsActiveFlag_RXNE*
	STOPF	*LL_I2C_IsActiveFlag_STOP*
	TC	*LL_I2C_IsActiveFlag_TC*
	TCR	*LL_I2C_IsActiveFlag_TCR*
	TIMEOUT	*LL_I2C_IsActiveSMBusFlag_TIMEOUT*
	TXE	*LL_I2C_ClearFlag_TXE*
		LL_I2C_IsActiveFlag_TXE
	TXIS	*LL_I2C_IsActiveFlag_TXIS*
OAR1	OA1	*LL_I2C_SetOwnAddress1*
	OA1EN	*LL_I2C_DisableOwnAddress1*
		LL_I2C_EnableOwnAddress1
		LL_I2C_IsEnabledOwnAddress1
	OA1MODE	*LL_I2C_SetOwnAddress1*
OAR2	OA2	*LL_I2C_SetOwnAddress2*
	OA2EN	*LL_I2C_DisableOwnAddress2*
		LL_I2C_EnableOwnAddress2
		LL_I2C_IsEnabledOwnAddress2
	OA2MSK	*LL_I2C_SetOwnAddress2*
PECR	PEC	*LL_I2C_GetSMBusPEC*
RXDR	RXDATA	*LL_I2C_DMA_GetRegAddr*
		LL_I2C_ReceiveData8
TIMEOUTR	TEXTEN	*LL_I2C_DisableSMBusTimeout*
		LL_I2C_EnableSMBusTimeout
		LL_I2C_IsEnabledSMBusTimeout

Register	Field	Function
	TIDLE	*LL_I2C_ConfigSMBusTimeout*
		LL_I2C_GetSMBusTimeoutAMode
		LL_I2C_SetSMBusTimeoutAMode
	TIMEOUTA	*LL_I2C_ConfigSMBusTimeout*
		LL_I2C_GetSMBusTimeoutA
		LL_I2C_SetSMBusTimeoutA
	TIMEOUTB	*LL_I2C_ConfigSMBusTimeout*
		LL_I2C_GetSMBusTimeoutB
		LL_I2C_SetSMBusTimeoutB
	TIMOUTEN	*LL_I2C_DisableSMBusTimeout*
		LL_I2C_EnableSMBusTimeout
		LL_I2C_IsEnabledSMBusTimeout
TIMINGR	PRESC	*LL_I2C_GetTimingPrescaler*
	SCLDEL	*LL_I2C_GetDataSetupTime*
	SCLH	*LL_I2C_GetClockHighPeriod*
	SCLL	*LL_I2C_GetClockLowPeriod*
	SDADEL	*LL_I2C_GetDataHoldTime*
	TIMINGR	*LL_I2C_SetTiming*
TXDR	TXDATA	*LL_I2C_DMA_GetRegAddr*
		LL_I2C_TransmitData8

100.14 IWDG

Table 39: Correspondence between IWDG registers and IWDG low-layer driver functions

Register	Field	Function
KR	KEY	*LL_IWDG_DisableWriteAccess*
		LL_IWDG_Enable
		LL_IWDG_EnableWriteAccess
		LL_IWDG_ReloadCounter
PR	PR	*LL_IWDG_GetPrescaler*
		LL_IWDG_SetPrescaler
RLR	RL	*LL_IWDG_GetReloadCounter*
		LL_IWDG_SetReloadCounter
SR	PVU	*LL_IWDG_IsActiveFlag_PVU*
		LL_IWDG_IsReady
	RVU	*LL_IWDG_IsActiveFlag_RVU*
		LL_IWDG_IsReady

Register	Field	Function
	WVU	*LL_IWDG_IsActiveFlag_WVU*
		LL_IWDG_IsReady
WINR	WIN	*LL_IWDG_GetWindow*
		LL_IWDG_SetWindow

100.15 LPTIM

Table 40: Correspondence between LPTIM registers and LPTIM low-layer driver functions

Register	Field	Function
ARR	ARR	*LL_LPTIM_GetAutoReload*
		LL_LPTIM_SetAutoReload
CFGR	CKFLT	*LL_LPTIM_ConfigClock*
		LL_LPTIM_GetClockFilter
	CKPOL	*LL_LPTIM_ConfigClock*
		LL_LPTIM_GetClockPolarity
		LL_LPTIM_GetEncoderMode
		LL_LPTIM_SetEncoderMode
	CKSEL	*LL_LPTIM_GetClockSource*
		LL_LPTIM_SetClockSource
	COUNTMODE	*LL_LPTIM_GetCounterMode*
		LL_LPTIM_SetCounterMode
	ENC	*LL_LPTIM_DisableEncoderMode*
		LL_LPTIM_EnableEncoderMode
		LL_LPTIM_IsEnabledEncoderMode
	PRELOAD	*LL_LPTIM_GetUpdateMode*
		LL_LPTIM_SetUpdateMode
	PRESC	*LL_LPTIM_GetPrescaler*
		LL_LPTIM_SetPrescaler
	TIMOUT	*LL_LPTIM_DisableTimeout*
		LL_LPTIM_EnableTimeout
		LL_LPTIM_IsEnabledTimeout
	TRGFLT	*LL_LPTIM_ConfigTrigger*
		LL_LPTIM_GetTriggerFilter
	TRIGEN	*LL_LPTIM_ConfigTrigger*
		LL_LPTIM_GetTriggerPolarity
		LL_LPTIM_TrigSw
	TRIGSEL	*LL_LPTIM_ConfigTrigger*

Register	Field	Function
		LL_LPTIM_GetTriggerSource
	WAVE	*LL_LPTIM_ConfigOutput*
		LL_LPTIM_GetWaveform
		LL_LPTIM_SetWaveform
	WAVPOL	*LL_LPTIM_ConfigOutput*
		LL_LPTIM_GetPolarity
		LL_LPTIM_SetPolarity
CMP	CMP	*LL_LPTIM_GetCompare*
		LL_LPTIM_SetCompare
CNT	CNT	*LL_LPTIM_GetCounter*
CR	CNTSTRT	*LL_LPTIM_StartCounter*
	ENABLE	*LL_LPTIM_Disable*
		LL_LPTIM_Enable
		LL_LPTIM_IsEnabled
	SNGSTRT	*LL_LPTIM_StartCounter*
ICR	ARRMCF	*LL_LPTIM_ClearFLAG_ARRM*
	ARROKCF	*LL_LPTIM_ClearFlag_ARROK*
	CMPMCF	*LL_LPTIM_ClearFLAG_CMPM*
	CMPOKCF	*LL_LPTIM_ClearFlag_CMPOK*
	DOWNCF	*LL_LPTIM_ClearFlag_DOWN*
	EXTTRIGCF	*LL_LPTIM_ClearFlag_EXTTRIG*
	UPCF	*LL_LPTIM_ClearFlag_UP*
IER	ARRMIE	*LL_LPTIM_DisableIT_ARRM*
		LL_LPTIM_EnableIT_ARRM
		LL_LPTIM_IsEnabledIT_ARRM
	ARROKIE	*LL_LPTIM_DisableIT_ARROK*
		LL_LPTIM_EnableIT_ARROK
		LL_LPTIM_IsEnabledIT_ARROK
	CMPMIE	*LL_LPTIM_DisableIT_CMPM*
		LL_LPTIM_EnableIT_CMPM
		LL_LPTIM_IsEnabledIT_CMPM
	CMPOKIE	*LL_LPTIM_DisableIT_CMPOK*
		LL_LPTIM_EnableIT_CMPOK
		LL_LPTIM_IsEnabledIT_CMPOK
	DOWNIE	*LL_LPTIM_DisableIT_DOWN*
		LL_LPTIM_EnableIT_DOWN

Register	Field	Function
		LL_LPTIM_IsEnabledIT_DOWN
	EXTTRIGIE	LL_LPTIM_DisableIT_EXTTRIG
		LL_LPTIM_EnableIT_EXTTRIG
		LL_LPTIM_IsEnabledIT_EXTTRIG
	UPIE	LL_LPTIM_DisableIT_UP
		LL_LPTIM_EnableIT_UP
		LL_LPTIM_IsEnabledIT_UP
ISR	ARRM	LL_LPTIM_IsActiveFlag_ARRM
	ARROK	LL_LPTIM_IsActiveFlag_ARROK
	CMPM	LL_LPTIM_IsActiveFlag_CMPM
	CMPOK	LL_LPTIM_IsActiveFlag_CMPOK
	DOWN	LL_LPTIM_IsActiveFlag_DOWN
	EXTTRIG	LL_LPTIM_IsActiveFlag_EXTTRIG
	UP	LL_LPTIM_IsActiveFlag_UP
OR	OR_0	LL_LPTIM_SetInput1Src
		LL_LPTIM_SetInput2Src
	OR_1	LL_LPTIM_SetInput1Src

100.16 LPUART

Table 41: Correspondence between LPUART registers and LPUART low-layer driver functions

Register	Field	Function
BRR	BRR	LL_LPUART_GetBaudRate
		LL_LPUART_SetBaudRate
CR1	CMIE	LL_LPUART_DisableIT_CM
		LL_LPUART_EnableIT_CM
		LL_LPUART_IsEnabledIT_CM
	DEAT	LL_LPUART_GetDEAssertionTime
		LL_LPUART_SetDEAssertionTime
	DEDT	LL_LPUART_GetDEDeassertionTime
		LL_LPUART_SetDEDeassertionTime
	FIFOEN	LL_LPUART_DisableFIFO
		LL_LPUART_EnableFIFO
		LL_LPUART_IsEnabledFIFO
	IDLEIE	LL_LPUART_DisableIT_IDLE
		LL_LPUART_EnableIT_IDLE
		LL_LPUART_IsEnabledIT_IDLE

Register	Field	Function
	M	*LL_LPUART_ConfigCharacter*
		LL_LPUART_GetDataWidth
		LL_LPUART_SetDataWidth
	MME	*LL_LPUART_DisableMuteMode*
		LL_LPUART_EnableMuteMode
		LL_LPUART_IsEnabledMuteMode
	PCE	*LL_LPUART_ConfigCharacter*
		LL_LPUART_GetParity
		LL_LPUART_SetParity
	PEIE	*LL_LPUART_DisableIT_PE*
		LL_LPUART_EnableIT_PE
		LL_LPUART_IsEnabledIT_PE
	PS	*LL_LPUART_ConfigCharacter*
		LL_LPUART_GetParity
		LL_LPUART_SetParity
	RE	*LL_LPUART_DisableDirectionRx*
		LL_LPUART_EnableDirectionRx
		LL_LPUART_GetTransferDirection
		LL_LPUART_SetTransferDirection
	RXFFIE	*LL_LPUART_DisableIT_RXFF*
		LL_LPUART_EnableIT_RXFF
		LL_LPUART_IsEnabledIT_RXFF
	RXNEIE_RXFNEIE	*LL_LPUART_DisableIT_RXNE_RXFNE*
		LL_LPUART_EnableIT_RXNE_RXFNE
		LL_LPUART_IsEnabledIT_RXNE_RXFNE
	TCIE	*LL_LPUART_DisableIT_TC*
		LL_LPUART_EnableIT_TC
		LL_LPUART_IsEnabledIT_TC
	TE	*LL_LPUART_DisableDirectionTx*
		LL_LPUART_EnableDirectionTx
		LL_LPUART_GetTransferDirection
		LL_LPUART_SetTransferDirection
	TXEIE_TXFNFIE	*LL_LPUART_DisableIT_TXE_TXFNF*
		LL_LPUART_EnableIT_TXE_TXFNF
		LL_LPUART_IsEnabledIT_TXE_TXFNF
	TXFEIE	*LL_LPUART_DisableIT_TXFE*

Register	Field	Function
		LL_LPUART_EnableIT_TXFE
		LL_LPUART_IsEnabledIT_TXFE
	UE	*LL_LPUART_Disable*
		LL_LPUART_Enable
		LL_LPUART_IsEnabled
	UESM	*LL_LPUART_DisableInStopMode*
		LL_LPUART_EnableInStopMode
		LL_LPUART_IsEnabledInStopMode
	WAKE	*LL_LPUART_GetWakeUpMethod*
		LL_LPUART_SetWakeUpMethod
CR2	ADD	*LL_LPUART_ConfigNodeAddress*
		LL_LPUART_GetNodeAddress
	ADDM7	*LL_LPUART_ConfigNodeAddress*
		LL_LPUART_GetNodeAddressLen
	DATAINV	*LL_LPUART_GetBinaryDataLogic*
		LL_LPUART_SetBinaryDataLogic
	MSBFIRST	*LL_LPUART_GetTransferBitOrder*
		LL_LPUART_SetTransferBitOrder
	RXINV	*LL_LPUART_GetRXPinLevel*
		LL_LPUART_SetRXPinLevel
	STOP	*LL_LPUART_ConfigCharacter*
		LL_LPUART_GetStopBitsLength
		LL_LPUART_SetStopBitsLength
	SWAP	*LL_LPUART_GetTXRXSwap*
		LL_LPUART_SetTXRXSwap
	TXINV	*LL_LPUART_GetTXPinLevel*
		LL_LPUART_SetTXPinLevel
CR3	CTSE	*LL_LPUART_DisableCTSHWFlowCtrl*
		LL_LPUART_EnableCTSHWFlowCtrl
		LL_LPUART_GetHWFlowCtrl
		LL_LPUART_SetHWFlowCtrl
	CTSIE	*LL_LPUART_DisableIT_CTS*
		LL_LPUART_EnableIT_CTS
		LL_LPUART_IsEnabledIT_CTS
	DDRE	*LL_LPUART_DisableDMADeactOnRxErr*
		LL_LPUART_EnableDMADeactOnRxErr

Register	Field	Function
		LL_LPUART_IsEnabledDMADeactOnRxErr
	DEM	*LL_LPUART_DisableDEMode*
		LL_LPUART_EnableDEMode
		LL_LPUART_IsEnabledDEMode
	DEP	*LL_LPUART_GetDESignalPolarity*
		LL_LPUART_SetDESignalPolarity
	DMAR	*LL_LPUART_DisableDMAReq_RX*
		LL_LPUART_EnableDMAReq_RX
		LL_LPUART_IsEnabledDMAReq_RX
	DMAT	*LL_LPUART_DisableDMAReq_TX*
		LL_LPUART_EnableDMAReq_TX
		LL_LPUART_IsEnabledDMAReq_TX
	EIE	*LL_LPUART_DisableIT_ERROR*
		LL_LPUART_EnableIT_ERROR
		LL_LPUART_IsEnabledIT_ERROR
	HDSEL	*LL_LPUART_DisableHalfDuplex*
		LL_LPUART_EnableHalfDuplex
		LL_LPUART_IsEnabledHalfDuplex
	OVRDIS	*LL_LPUART_DisableOverrunDetect*
		LL_LPUART_EnableOverrunDetect
		LL_LPUART_IsEnabledOverrunDetect
	RTSE	*LL_LPUART_DisableRTSHWFlowCtrl*
		LL_LPUART_EnableRTSHWFlowCtrl
		LL_LPUART_GetHWFlowCtrl
		LL_LPUART_SetHWFlowCtrl
	RXFTCFG	*LL_LPUART_ConfigFIFOsThreshold*
		LL_LPUART_GetRXFIFOThreshold
		LL_LPUART_SetRXFIFOThreshold
	RXFTIE	*LL_LPUART_DisableIT_RXFT*
		LL_LPUART_EnableIT_RXFT
		LL_LPUART_IsEnabledIT_RXFT
	TXFTCFG	*LL_LPUART_ConfigFIFOsThreshold*
		LL_LPUART_GetTXFIFOThreshold
		LL_LPUART_SetTXFIFOThreshold
	TXFTIE	*LL_LPUART_DisableIT_TXFT*
		LL_LPUART_EnableIT_TXFT

Register	Field	Function
		LL_LPUART_IsEnabledIT_TXFT
	WUFIE	*LL_LPUART_DisableIT_WKUP*
		LL_LPUART_EnableIT_WKUP
		LL_LPUART_IsEnabledIT_WKUP
	WUS	*LL_LPUART_GetWKUPType*
		LL_LPUART_SetWKUPType
ICR	CMCF	*LL_LPUART_ClearFlag_CM*
	CTSCF	*LL_LPUART_ClearFlag_nCTS*
	FECF	*LL_LPUART_ClearFlag_FE*
	IDLECF	*LL_LPUART_ClearFlag_IDLE*
	NCF	*LL_LPUART_ClearFlag_NE*
	ORECF	*LL_LPUART_ClearFlag_ORE*
	PECF	*LL_LPUART_ClearFlag_PE*
	TCCF	*LL_LPUART_ClearFlag_TC*
	TXFECF	*LL_LPUART_ClearFlag_TXFE*
	WUCF	*LL_LPUART_ClearFlag_WKUP*
ISR	BUSY	*LL_LPUART_IsActiveFlag_BUSY*
	CMF	*LL_LPUART_IsActiveFlag_CM*
	CTS	*LL_LPUART_IsActiveFlag_CTS*
	CTSIF	*LL_LPUART_IsActiveFlag_nCTS*
	FE	*LL_LPUART_IsActiveFlag_FE*
	IDLE	*LL_LPUART_IsActiveFlag_IDLE*
	NE	*LL_LPUART_IsActiveFlag_NE*
	ORE	*LL_LPUART_IsActiveFlag_ORE*
	PE	*LL_LPUART_IsActiveFlag_PE*
	REACK	*LL_LPUART_IsActiveFlag_REACK*
	RWU	*LL_LPUART_IsActiveFlag_RWU*
	RXFF	*LL_LPUART_IsActiveFlag_RXFF*
	RXFT	*LL_LPUART_IsActiveFlag_RXFT*
	RXNE_RXFNE	*LL_LPUART_IsActiveFlag_RXNE_RXFNE*
	SBKF	*LL_LPUART_IsActiveFlag_SBK*
	TC	*LL_LPUART_IsActiveFlag_TC*
	TEACK	*LL_LPUART_IsActiveFlag_TEACK*
	TXE_TXFNF	*LL_LPUART_IsActiveFlag_TXE_TXFNF*
	TXFE	*LL_LPUART_IsActiveFlag_TXFE*
	TXFT	*LL_LPUART_IsActiveFlag_TXFT*

Register	Field	Function
	WUF	*LL_LPUART_IsActiveFlag_WKUP*
PRESC	PRESCALER	*LL_LPUART_GetPrescaler*
		LL_LPUART_SetPrescaler
RDR	RDR	*LL_LPUART_DMA_GetRegAddr*
		LL_LPUART_ReceiveData8
		LL_LPUART_ReceiveData9
RQR	MMRQ	*LL_LPUART_RequestEnterMuteMode*
	RXFRQ	*LL_LPUART_RequestRxDataFlush*
	SBKRQ	*LL_LPUART_RequestBreakSending*
TDR	TDR	*LL_LPUART_DMA_GetRegAddr*
		LL_LPUART_TransmitData8
		LL_LPUART_TransmitData9

100.17 OPAMP

Table 42: Correspondence between OPAMP registers and OPAMP low-layer driver functions

Register	Field	Function
CSR	CALON	*LL_OPAMP_GetMode*
		LL_OPAMP_SetMode
	CALOUT	*LL_OPAMP_IsCalibrationOutputSet*
	CALSEL	*LL_OPAMP_GetCalibrationSelection*
		LL_OPAMP_SetCalibrationSelection
	OPALPM	*LL_OPAMP_GetPowerMode*
		LL_OPAMP_SetPowerMode
	OPAMODE	*LL_OPAMP_GetFunctionalMode*
		LL_OPAMP_SetFunctionalMode
	OPAMPXEN	*LL_OPAMP_Disable*
		LL_OPAMP_Enable
		LL_OPAMP_IsEnabled
	OPARANGE	*LL_OPAMP_GetCommonPowerRange*
		LL_OPAMP_SetCommonPowerRange
	PGGAIN	*LL_OPAMP_GetPGAGain*
		LL_OPAMP_SetPGAGain
	USERTRIM	*LL_OPAMP_GetTrimmingMode*
		LL_OPAMP_SetTrimmingMode
	VMSEL	*LL_OPAMP_GetInputInverting*
		LL_OPAMP_SetInputInverting

Register	Field	Function
	VPSEL	LL_OPAMP_GetInputNonInverting
		LL_OPAMP_SetInputNonInverting
LPOTR	TRIMLPOFFSETN	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue
	TRIMLPOFFSETP	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue
OTR	TRIMOFFSETN	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue
	TRIMOFFSETP	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue

100.18 PWR

Table 43: Correspondence between PWR registers and PWR low-layer driver functions

Register	Field	Function
CR1	DBP	LL_PWR_DisableBkUpAccess
		LL_PWR_EnableBkUpAccess
		LL_PWR_IsEnabledBkUpAccess
	LPMS	LL_PWR_GetPowerMode
		LL_PWR_SetPowerMode
	LPR	LL_PWR_DisableLowPowerRunMode
		LL_PWR_EnableLowPowerRunMode
		LL_PWR_EnterLowPowerRunMode
		LL_PWR_ExitLowPowerRunMode
		LL_PWR_IsEnabledLowPowerRunMode
	RRSTP	LL_PWR_DisableSRAM3Retention
		LL_PWR_EnableSRAM3Retention
		LL_PWR_IsEnabledSRAM3Retention
	VOS	LL_PWR_GetRegulVoltageScaling
		LL_PWR_SetRegulVoltageScaling
CR2	IOSV	LL_PWR_DisableVddIO2
		LL_PWR_EnableVddIO2
		LL_PWR_IsEnabledVddIO2
	PLS	LL_PWR_GetPVDLevel
		LL_PWR_SetPVDLevel
	PVDE	LL_PWR_DisablePVD
		LL_PWR_EnablePVD

Register	Field	Function
		LL_PWR_IsEnabledPVD
	PVME1	*LL_PWR_DisablePVM*
		LL_PWR_EnablePVM
		LL_PWR_IsEnabledPVM
	PVME2	*LL_PWR_DisablePVM*
		LL_PWR_EnablePVM
		LL_PWR_IsEnabledPVM
	PVME3	*LL_PWR_DisablePVM*
		LL_PWR_EnablePVM
		LL_PWR_IsEnabledPVM
	PVME4	*LL_PWR_DisablePVM*
		LL_PWR_EnablePVM
		LL_PWR_IsEnabledPVM
	USV	*LL_PWR_DisableVddUSB*
		LL_PWR_EnableVddUSB
		LL_PWR_IsEnabledVddUSB
CR3	APC	*LL_PWR_DisablePUPDCfg*
		LL_PWR_EnablePUPDCfg
		LL_PWR_IsEnabledPUPDCfg
	DSIPDEN	*LL_PWR_DisableDSIPinsPDActivation*
		LL_PWR_DisableDSIPullDown
		LL_PWR_EnableDSIPinsPDActivation
		LL_PWR_EnableDSIPullDown
		LL_PWR_IsEnabledDSIPinsPDActivation
		LL_PWR_IsEnabledDSIPullDown
	EIWF	*LL_PWR_DisableInternWU*
		LL_PWR_EnableInternWU
		LL_PWR_IsEnabledInternWU
	EWUP1	*LL_PWR_DisableWakeUpPin*
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP2	*LL_PWR_DisableWakeUpPin*
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP3	*LL_PWR_DisableWakeUpPin*
		LL_PWR_EnableWakeUpPin

Register	Field	Function
		LL_PWR_IsEnabledWakeUpPin
	EWUP4	*LL_PWR_DisableWakeUpPin*
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWUP5	*LL_PWR_DisableWakeUpPin*
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	RRS	*LL_PWR_DisableSRAM2Retention*
		LL_PWR_EnableSRAM2Retention
		LL_PWR_IsEnabledSRAM2Retention
CR4	VBE	*LL_PWR_DisableBatteryCharging*
		LL_PWR_EnableBatteryCharging
		LL_PWR_IsEnabledBatteryCharging
	VBRS	*LL_PWR_GetBattChargResistor*
		LL_PWR_SetBattChargResistor
	WP1	*LL_PWR_IsWakeUpPinPolarityLow*
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WP2	*LL_PWR_IsWakeUpPinPolarityLow*
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WP3	*LL_PWR_IsWakeUpPinPolarityLow*
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WP4	*LL_PWR_IsWakeUpPinPolarityLow*
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WP5	*LL_PWR_IsWakeUpPinPolarityLow*
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
CR5	R1MODE	*LL_PWR_DisableRange1BoostMode*
		LL_PWR_EnableRange1BoostMode
		LL_PWR_IsEnabledRange1BoostMode
PDCRA	PD0-15	*LL_PWR_DisableGPIOPullDown*
		LL_PWR_EnableGPIOPullDown
		LL_PWR_IsEnabledGPIOPullDown

Register	Field	Function
PDCRB	PD0-15	*LL_PWR_DisableGPIOPullDown*
		LL_PWR_EnableGPIOPullDown
		LL_PWR_IsEnabledGPIOPullDown
PDCRC	PD0-15	*LL_PWR_DisableGPIOPullDown*
		LL_PWR_EnableGPIOPullDown
		LL_PWR_IsEnabledGPIOPullDown
PDCRD	PD0-15	*LL_PWR_DisableGPIOPullDown*
		LL_PWR_EnableGPIOPullDown
		LL_PWR_IsEnabledGPIOPullDown
PDCRE	PD0-15	*LL_PWR_DisableGPIOPullDown*
		LL_PWR_EnableGPIOPullDown
		LL_PWR_IsEnabledGPIOPullDown
PDCRF	PD0-15	*LL_PWR_DisableGPIOPullDown*
		LL_PWR_EnableGPIOPullDown
		LL_PWR_IsEnabledGPIOPullDown
PDCRG	PD0-15	*LL_PWR_DisableGPIOPullDown*
		LL_PWR_EnableGPIOPullDown
		LL_PWR_IsEnabledGPIOPullDown
PDCRH	PD0-15	*LL_PWR_DisableGPIOPullDown*
		LL_PWR_EnableGPIOPullDown
		LL_PWR_IsEnabledGPIOPullDown
PDCRI	PD0-11	*LL_PWR_DisableGPIOPullDown*
		LL_PWR_EnableGPIOPullDown
		LL_PWR_IsEnabledGPIOPullDown
PUCRA	PU0-15	*LL_PWR_DisableGPIOPullUp*
		LL_PWR_EnableGPIOPullUp
		LL_PWR_IsEnabledGPIOPullUp
PUCRB	PU0-15	*LL_PWR_DisableGPIOPullUp*
		LL_PWR_EnableGPIOPullUp
		LL_PWR_IsEnabledGPIOPullUp
PUCRC	PU0-15	*LL_PWR_DisableGPIOPullUp*
		LL_PWR_EnableGPIOPullUp
		LL_PWR_IsEnabledGPIOPullUp
PUCRD	PU0-15	*LL_PWR_DisableGPIOPullUp*
		LL_PWR_EnableGPIOPullUp
		LL_PWR_IsEnabledGPIOPullUp

Register	Field	Function
PUCRE	PU0-15	LL_PWR_DisableGPIOPullUp
		LL_PWR_EnableGPIOPullUp
		LL_PWR_IsEnabledGPIOPullUp
PUCRF	PU0-15	LL_PWR_DisableGPIOPullUp
		LL_PWR_EnableGPIOPullUp
		LL_PWR_IsEnabledGPIOPullUp
PUCRG	PU0-15	LL_PWR_DisableGPIOPullUp
		LL_PWR_EnableGPIOPullUp
		LL_PWR_IsEnabledGPIOPullUp
PUCRH	PU0-15	LL_PWR_DisableGPIOPullUp
		LL_PWR_EnableGPIOPullUp
		LL_PWR_IsEnabledGPIOPullUp
PUCRI	PU0-11	LL_PWR_DisableGPIOPullUp
		LL_PWR_EnableGPIOPullUp
		LL_PWR_IsEnabledGPIOPullUp
SCR	CSBF	LL_PWR_ClearFlag_SB
	CWUF	LL_PWR_ClearFlag_WU
	CWUF1	LL_PWR_ClearFlag_WU1
	CWUF2	LL_PWR_ClearFlag_WU2
	CWUF3	LL_PWR_ClearFlag_WU3
	CWUF4	LL_PWR_ClearFlag_WU4
	CWUF5	LL_PWR_ClearFlag_WU5
SR1	SBF	LL_PWR_IsActiveFlag_SB
	WUF1	LL_PWR_IsActiveFlag_WU1
	WUF2	LL_PWR_IsActiveFlag_WU2
	WUF3	LL_PWR_IsActiveFlag_WU3
	WUF4	LL_PWR_IsActiveFlag_WU4
	WUF5	LL_PWR_IsActiveFlag_WU5
	WUFI	LL_PWR_IsActiveFlag_InternWU
SR2	PVDO	LL_PWR_IsActiveFlag_PVDO
	PVMO1	LL_PWR_IsActiveFlag_PVMO1
	PVMO2	LL_PWR_IsActiveFlag_PVMO2
	PVMO3	LL_PWR_IsActiveFlag_PVMO3
	PVMO4	LL_PWR_IsActiveFlag_PVMO4
	REGLPF	LL_PWR_IsActiveFlag_REGLPF
	REGLPS	LL_PWR_IsActiveFlag_REGLPS

Register	Field	Function
	VOSF	*LL_PWR_IsActiveFlag_VOS*

100.19 RCC

Table 44: Correspondence between RCC registers and RCC low-layer driver functions

Register	Field	Function
BDCR	BDRST	*LL_RCC_ForceBackupDomainReset*
		LL_RCC_ReleaseBackupDomainReset
	LSCOEN	*LL_RCC_LSCO_Disable*
		LL_RCC_LSCO_Enable
	LSCOSEL	*LL_RCC_LSCO_GetSource*
		LL_RCC_LSCO_SetSource
	LSEBYP	*LL_RCC_LSE_DisableBypass*
		LL_RCC_LSE_EnableBypass
	LSECSSD	*LL_RCC_LSE_IsCSSDetected*
	LSECSSON	*LL_RCC_LSE_DisableCSS*
		LL_RCC_LSE_EnableCSS
	LSEDRV	*LL_RCC_LSE_GetDriveCapability*
		LL_RCC_LSE_SetDriveCapability
	LSEON	*LL_RCC_LSE_Disable*
		LL_RCC_LSE_Enable
	LSERDY	*LL_RCC_LSE_IsReady*
	RTCEN	*LL_RCC_DisableRTC*
		LL_RCC_EnableRTC
		LL_RCC_IsEnabledRTC
	RTCSEL	*LL_RCC_GetRTCClockSource*
		LL_RCC_SetRTCClockSource
CCIPR	ADCSEL	*LL_RCC_GetADCClockSource*
		LL_RCC_SetADCClockSource
	CLK48SEL	*LL_RCC_GetRNGClockSource*
		LL_RCC_GetUSBClockSource
		LL_RCC_SetRNGClockSource
		LL_RCC_SetUSBClockSource
	I2CxSEL	*LL_RCC_GetI2CClockSource*
		LL_RCC_SetI2CClockSource
	LPTIMxSEL	*LL_RCC_GetLPTIMClockSource*
		LL_RCC_SetLPTIMClockSource

Register	Field	Function
	LPUART1SEL	*LL_RCC_GetLPUARTClockSource*
		LL_RCC_SetLPUARTClockSource
	SAIxSEL	*LL_RCC_GetSAIClockSource*
	UARTxSEL	*LL_RCC_GetUARTClockSource*
		LL_RCC_SetUARTClockSource
	USARTxSEL	*LL_RCC_GetUSARTClockSource*
		LL_RCC_SetUSARTClockSource
CCIPR2	ADFSDM1SEL	*LL_RCC_GetDFSDMAudioClockSource*
		LL_RCC_SetDFSDMAudioClockSource
	DFSDM1SEL	*LL_RCC_GetDFSDMClockSource*
		LL_RCC_SetDFSDMClockSource
	DSISEL	*LL_RCC_GetDSIClockSource*
		LL_RCC_SetDSIClockSource
	OSPISEL	*LL_RCC_GetOCTOSPIClockSource*
		LL_RCC_SetOCTOSPIClockSource
	PLLSAI2DIVR	*LL_RCC_GetLTDCClockSource*
		LL_RCC_PLLSAI2_ConfigDomain_LTDC
		LL_RCC_PLLSAI2_GetDIVR
		LL_RCC_SetLTDCClockSource
	SAIxSEL	*LL_RCC_SetSAIClockSource*
	SDMMCSEL	*LL_RCC_GetSDMMCClockSource*
		LL_RCC_SetSDMMCClockSource
CFGR	HPRE	*LL_RCC_GetAHBPrescaler*
		LL_RCC_SetAHBPrescaler
	MCOPRE	*LL_RCC_ConfigMCO*
	MCOSEL	*LL_RCC_ConfigMCO*
	PPRE1	*LL_RCC_GetAPB1Prescaler*
		LL_RCC_SetAPB1Prescaler
	PPRE2	*LL_RCC_GetAPB2Prescaler*
		LL_RCC_SetAPB2Prescaler
	STOPWUCK	*LL_RCC_GetClkAfterWakeFromStop*
		LL_RCC_SetClkAfterWakeFromStop
	SW	*LL_RCC_SetSysClkSource*
	SWS	*LL_RCC_GetSysClkSource*
CICR	CSSC	*LL_RCC_ClearFlag_HSECSS*
	HSERDYC	*LL_RCC_ClearFlag_HSERDY*

Register	Field	Function
	HSI48RDYC	*LL_RCC_ClearFlag_HSI48RDY*
	HSIRDYC	*LL_RCC_ClearFlag_HSIRDY*
	LSECSSC	*LL_RCC_ClearFlag_LSECSS*
	LSERDYC	*LL_RCC_ClearFlag_LSERDY*
	LSIRDYC	*LL_RCC_ClearFlag_LSIRDY*
	MSIRDYC	*LL_RCC_ClearFlag_MSIRDY*
	PLLRDYC	*LL_RCC_ClearFlag_PLLRDY*
	PLLSAI1RDYC	*LL_RCC_ClearFlag_PLLSAI1RDY*
	PLLSAI2RDYC	*LL_RCC_ClearFlag_PLLSAI2RDY*
CIER	HSERDYIE	*LL_RCC_DisableIT_HSERDY*
		LL_RCC_EnableIT_HSERDY
		LL_RCC_IsEnabledIT_HSERDY
	HSI48RDYIE	*LL_RCC_DisableIT_HSI48RDY*
		LL_RCC_EnableIT_HSI48RDY
		LL_RCC_IsEnabledIT_HSI48RDY
	HSIRDYIE	*LL_RCC_DisableIT_HSIRDY*
		LL_RCC_EnableIT_HSIRDY
		LL_RCC_IsEnabledIT_HSIRDY
	LSECSSIE	*LL_RCC_DisableIT_LSECSS*
		LL_RCC_EnableIT_LSECSS
		LL_RCC_IsEnabledIT_LSECSS
	LSERDYIE	*LL_RCC_DisableIT_LSERDY*
		LL_RCC_EnableIT_LSERDY
		LL_RCC_IsEnabledIT_LSERDY
	LSIRDYIE	*LL_RCC_DisableIT_LSIRDY*
		LL_RCC_EnableIT_LSIRDY
		LL_RCC_IsEnabledIT_LSIRDY
	MSIRDYIE	*LL_RCC_DisableIT_MSIRDY*
		LL_RCC_EnableIT_MSIRDY
		LL_RCC_IsEnabledIT_MSIRDY
	PLLRDYIE	*LL_RCC_DisableIT_PLLRDY*
		LL_RCC_EnableIT_PLLRDY
		LL_RCC_IsEnabledIT_PLLRDY
	PLLSAI1RDYIE	*LL_RCC_DisableIT_PLLSAI1RDY*
		LL_RCC_EnableIT_PLLSAI1RDY
		LL_RCC_IsEnabledIT_PLLSAI1RDY

Register	Field	Function
	PLLSAI2RDYIE	*LL_RCC_DisableIT_PLLSAI2RDY*
		LL_RCC_EnableIT_PLLSAI2RDY
		LL_RCC_IsEnabledIT_PLLSAI2RDY
CIFR	CSSF	*LL_RCC_IsActiveFlag_HSECSS*
	HSERDYF	*LL_RCC_IsActiveFlag_HSERDY*
	HSIRDYF	*LL_RCC_IsActiveFlag_HSIRDY*
	LSECSSF	*LL_RCC_IsActiveFlag_LSECSS*
	LSERDYF	*LL_RCC_IsActiveFlag_LSERDY*
	LSIRDYF	*LL_RCC_IsActiveFlag_LSIRDY*
	MSIRDYF	*LL_RCC_IsActiveFlag_MSIRDY*
	PLLRDYF	*LL_RCC_IsActiveFlag_PLLRDY*
	PLLSAI1RDYF	*LL_RCC_IsActiveFlag_PLLSAI1RDY*
	PLLSAI2RDYF	*LL_RCC_IsActiveFlag_PLLSAI2RDY*
CIR	HSI48RDYF	*LL_RCC_IsActiveFlag_HSI48RDY*
CR	CSSON	*LL_RCC_HSE_EnableCSS*
	HSEBYP	*LL_RCC_HSE_DisableBypass*
		LL_RCC_HSE_EnableBypass
	HSEON	*LL_RCC_HSE_Disable*
		LL_RCC_HSE_Enable
	HSERDY	*LL_RCC_HSE_IsReady*
	HSIASFS	*LL_RCC_HSI_DisableAutoFromStop*
		LL_RCC_HSI_EnableAutoFromStop
	HSIKERON	*LL_RCC_HSI_DisableInStopMode*
		LL_RCC_HSI_EnableInStopMode
	HSION	*LL_RCC_HSI_Disable*
		LL_RCC_HSI_Enable
	HSIRDY	*LL_RCC_HSI_IsReady*
	MSION	*LL_RCC_MSI_Disable*
		LL_RCC_MSI_Enable
	MSIPLLEN	*LL_RCC_MSI_DisablePLLMode*
		LL_RCC_MSI_EnablePLLMode
	MSIRANGE	*LL_RCC_MSI_GetRange*
		LL_RCC_MSI_SetRange
	MSIRDY	*LL_RCC_MSI_IsReady*
	MSIRGSEL	*LL_RCC_MSI_EnableRangeSelection*
		LL_RCC_MSI_IsEnabledRangeSelect

Register	Field	Function
	PLLON	*LL_RCC_PLL_Disable*
		LL_RCC_PLL_Enable
	PLLRDY	*LL_RCC_PLL_IsReady*
	PLLSAI1ON	*LL_RCC_PLLSAI1_Disable*
		LL_RCC_PLLSAI1_Enable
	PLLSAI1RDY	*LL_RCC_PLLSAI1_IsReady*
	PLLSAI2ON	*LL_RCC_PLLSAI2_Disable*
		LL_RCC_PLLSAI2_Enable
	PLLSAI2RDY	*LL_RCC_PLLSAI2_IsReady*
CRRCR	HSI48CAL	*LL_RCC_HSI48_GetCalibration*
	HSI48ON	*LL_RCC_HSI48_Disable*
		LL_RCC_HSI48_Enable
	HSI48RDY	*LL_RCC_HSI48_IsReady*
CSR	BORRSTF	*LL_RCC_IsActiveFlag_BORRST*
	FWRSTF	*LL_RCC_IsActiveFlag_FWRST*
	IWDGRSTF	*LL_RCC_IsActiveFlag_IWDGRST*
	LPWRRSTF	*LL_RCC_IsActiveFlag_LPWRRST*
	LSION	*LL_RCC_LSI_Disable*
		LL_RCC_LSI_Enable
	LSIRDY	*LL_RCC_LSI_IsReady*
	MSISRANGE	*LL_RCC_MSI_GetRangeAfterStandby*
		LL_RCC_MSI_SetRangeAfterStandby
	OBLRSTF	*LL_RCC_IsActiveFlag_OBLRST*
	PINRSTF	*LL_RCC_IsActiveFlag_PINRST*
	RMVF	*LL_RCC_ClearResetFlags*
	SFTRSTF	*LL_RCC_IsActiveFlag_SFTRST*
	WWDGRSTF	*LL_RCC_IsActiveFlag_WWDGRST*
ICSCR	HSICAL	*LL_RCC_HSI_GetCalibration*
	HSITRIM	*LL_RCC_HSI_GetCalibTrimming*
		LL_RCC_HSI_SetCalibTrimming
	MSICAL	*LL_RCC_MSI_GetCalibration*
	MSITRIM	*LL_RCC_MSI_GetCalibTrimming*
		LL_RCC_MSI_SetCalibTrimming
PLLCFGR	PLLM	*LL_RCC_PLL_ConfigDomain_48M*
		LL_RCC_PLL_ConfigDomain_SAI
		LL_RCC_PLL_ConfigDomain_SYS

Register	Field	Function
		LL_RCC_PLL_GetDivider
	PLLN	LL_RCC_PLL_ConfigDomain_48M
		LL_RCC_PLL_ConfigDomain_SAI
		LL_RCC_PLL_ConfigDomain_SYS
		LL_RCC_PLL_GetN
	PLLPDIV	LL_RCC_PLL_ConfigDomain_SAI
		LL_RCC_PLL_GetP
	PLLPEN	LL_RCC_PLL_DisableDomain_SAI
		LL_RCC_PLL_EnableDomain_SAI
	PLLQ	LL_RCC_PLL_ConfigDomain_48M
		LL_RCC_PLL_GetQ
	PLLQEN	LL_RCC_PLL_DisableDomain_48M
		LL_RCC_PLL_EnableDomain_48M
	PLLR	LL_RCC_PLL_ConfigDomain_SYS
		LL_RCC_PLL_GetR
	PLLREN	LL_RCC_PLL_DisableDomain_SYS
		LL_RCC_PLL_EnableDomain_SYS
	PLLSRC	LL_RCC_PLLSAI1_ConfigDomain_48M
		LL_RCC_PLLSAI1_ConfigDomain_ADC
		LL_RCC_PLLSAI1_ConfigDomain_SAI
		LL_RCC_PLLSAI2_ConfigDomain_DSI
		LL_RCC_PLLSAI2_ConfigDomain_LTDC
		LL_RCC_PLLSAI2_ConfigDomain_SAI
		LL_RCC_PLL_ConfigDomain_48M
		LL_RCC_PLL_ConfigDomain_SAI
		LL_RCC_PLL_ConfigDomain_SYS
		LL_RCC_PLL_GetMainSource
PLLSAI1CFGR	PLLSAI1M	LL_RCC_PLLSAI1_ConfigDomain_48M
		LL_RCC_PLLSAI1_ConfigDomain_ADC
		LL_RCC_PLLSAI1_ConfigDomain_SAI
		LL_RCC_PLLSAI1_GetDivider
	PLLSAI1N	LL_RCC_PLLSAI1_ConfigDomain_48M
		LL_RCC_PLLSAI1_ConfigDomain_ADC
		LL_RCC_PLLSAI1_ConfigDomain_SAI
		LL_RCC_PLLSAI1_GetN
	PLLSAI1PDIV	LL_RCC_PLLSAI1_ConfigDomain_SAI

Register	Field	Function
		LL_RCC_PLLSAI1_GetP
	PLLSAI1PEN	*LL_RCC_PLLSAI1_DisableDomain_SAI*
		LL_RCC_PLLSAI1_EnableDomain_SAI
	PLLSAI1Q	*LL_RCC_PLLSAI1_ConfigDomain_48M*
		LL_RCC_PLLSAI1_GetQ
	PLLSAI1QEN	*LL_RCC_PLLSAI1_DisableDomain_48M*
		LL_RCC_PLLSAI1_EnableDomain_48M
	PLLSAI1R	*LL_RCC_PLLSAI1_ConfigDomain_ADC*
		LL_RCC_PLLSAI1_GetR
	PLLSAI1REN	*LL_RCC_PLLSAI1_DisableDomain_ADC*
		LL_RCC_PLLSAI1_EnableDomain_ADC
PLLSAI2CFGR	PLLSAI2M	*LL_RCC_PLLSAI2_ConfigDomain_DSI*
		LL_RCC_PLLSAI2_ConfigDomain_LTDC
		LL_RCC_PLLSAI2_ConfigDomain_SAI
		LL_RCC_PLLSAI2_GetDivider
	PLLSAI2N	*LL_RCC_PLLSAI2_ConfigDomain_DSI*
		LL_RCC_PLLSAI2_ConfigDomain_LTDC
		LL_RCC_PLLSAI2_ConfigDomain_SAI
		LL_RCC_PLLSAI2_GetN
	PLLSAI2PDIV	*LL_RCC_PLLSAI2_ConfigDomain_SAI*
		LL_RCC_PLLSAI2_GetP
	PLLSAI2PEN	*LL_RCC_PLLSAI2_DisableDomain_SAI*
		LL_RCC_PLLSAI2_EnableDomain_SAI
	PLLSAI2Q	*LL_RCC_PLLSAI2_ConfigDomain_DSI*
		LL_RCC_PLLSAI2_GetQ
	PLLSAI2QEN	*LL_RCC_PLLSAI2_DisableDomain_DSI*
		LL_RCC_PLLSAI2_EnableDomain_DSI
	PLLSAI2R	*LL_RCC_PLLSAI2_ConfigDomain_LTDC*
		LL_RCC_PLLSAI2_GetR
	PLLSAI2REN	*LL_RCC_PLLSAI2_DisableDomain_LTDC*
		LL_RCC_PLLSAI2_EnableDomain_LTDC

100.20 RNG

Table 45: Correspondence between RNG registers and RNG low-layer driver functions

Register	Field	Function
CR	CED	*LL_RNG_DisableClkErrorDetect*
		LL_RNG_EnableClkErrorDetect
		LL_RNG_IsEnabledClkErrorDetect
	IE	*LL_RNG_DisableIT*
		LL_RNG_EnableIT
		LL_RNG_IsEnabledIT
	RNGEN	*LL_RNG_Disable*
		LL_RNG_Enable
		LL_RNG_IsEnabled
DR	RNDATA	*LL_RNG_ReadRandData32*
SR	CECS	*LL_RNG_IsActiveFlag_CECS*
	CEIS	*LL_RNG_ClearFlag_CEIS*
		LL_RNG_IsActiveFlag_CEIS
	DRDY	*LL_RNG_IsActiveFlag_DRDY*
	SECS	*LL_RNG_IsActiveFlag_SECS*
	SEIS	*LL_RNG_ClearFlag_SEIS*
		LL_RNG_IsActiveFlag_SEIS

100.21 RTC

Table 46: Correspondence between RTC registers and RTC low-layer driver functions

Register	Field	Function
ALRMAR	DT	*LL_RTC_ALMA_GetDay*
		LL_RTC_ALMA_SetDay
	DU	*LL_RTC_ALMA_GetDay*
		LL_RTC_ALMA_GetWeekDay
		LL_RTC_ALMA_SetDay
		LL_RTC_ALMA_SetWeekDay
	HT	*LL_RTC_ALMA_ConfigTime*
		LL_RTC_ALMA_GetHour
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetHour
	HU	*LL_RTC_ALMA_ConfigTime*
		LL_RTC_ALMA_GetHour
		LL_RTC_ALMA_GetTime

Register	Field	Function
		LL_RTC_ALMA_SetHour
	MNT	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetMinute
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetMinute
	MNU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetMinute
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetMinute
	MSK1	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK2	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK3	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK4	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	PM	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetTimeFormat
		LL_RTC_ALMA_SetTimeFormat
	ST	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetSecond
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetSecond
	SU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetSecond
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetSecond
	WDSEL	LL_RTC_ALMA_DisableWeekday
		LL_RTC_ALMA_EnableWeekday
ALRMASSR	MASKSS	LL_RTC_ALMA_GetSubSecondMask
		LL_RTC_ALMA_SetSubSecondMask
	SS	LL_RTC_ALMA_GetSubSecond
		LL_RTC_ALMA_SetSubSecond
ALRMBR	DT	LL_RTC_ALMB_GetDay
		LL_RTC_ALMB_SetDay

Register	Field	Function
	DU	*LL_RTC_ALMB_GetDay*
		LL_RTC_ALMB_GetWeekDay
		LL_RTC_ALMB_SetDay
		LL_RTC_ALMB_SetWeekDay
	HT	*LL_RTC_ALMB_ConfigTime*
		LL_RTC_ALMB_GetHour
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetHour
	HU	*LL_RTC_ALMB_ConfigTime*
		LL_RTC_ALMB_GetHour
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetHour
	MNT	*LL_RTC_ALMB_ConfigTime*
		LL_RTC_ALMB_GetMinute
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetMinute
	MNU	*LL_RTC_ALMB_ConfigTime*
		LL_RTC_ALMB_GetMinute
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetMinute
	MSK1	*LL_RTC_ALMB_GetMask*
		LL_RTC_ALMB_SetMask
	MSK2	*LL_RTC_ALMB_GetMask*
		LL_RTC_ALMB_SetMask
	MSK3	*LL_RTC_ALMB_GetMask*
		LL_RTC_ALMB_SetMask
	MSK4	*LL_RTC_ALMB_GetMask*
		LL_RTC_ALMB_SetMask
	PM	*LL_RTC_ALMB_ConfigTime*
		LL_RTC_ALMB_GetTimeFormat
		LL_RTC_ALMB_SetTimeFormat
	ST	*LL_RTC_ALMB_ConfigTime*
		LL_RTC_ALMB_GetSecond
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetSecond
	SU	*LL_RTC_ALMB_ConfigTime*

Register	Field	Function
		LL_RTC_ALMB_GetSecond
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetSecond
	WDSEL	LL_RTC_ALMB_DisableWeekday
		LL_RTC_ALMB_EnableWeekday
ALRMBSSR	MASKSS	LL_RTC_ALMB_GetSubSecondMask
		LL_RTC_ALMB_SetSubSecondMask
	SS	LL_RTC_ALMB_GetSubSecond
		LL_RTC_ALMB_SetSubSecond
BKPxR	BKP	LL_RTC_BAK_GetRegister
		LL_RTC_BAK_SetRegister
CALR	CALM	LL_RTC_CAL_GetMinus
		LL_RTC_CAL_SetMinus
	CALP	LL_RTC_CAL_IsPulseInserted
		LL_RTC_CAL_SetPulse
	CALW16	LL_RTC_CAL_GetPeriod
		LL_RTC_CAL_SetPeriod
	CALW8	LL_RTC_CAL_GetPeriod
		LL_RTC_CAL_SetPeriod
CR	ADD1H	LL_RTC_TIME_IncHour
	ALRAE	LL_RTC_ALMA_Disable
		LL_RTC_ALMA_Enable
	ALRAIE	LL_RTC_DisableIT_ALRA
		LL_RTC_EnableIT_ALRA
		LL_RTC_IsEnabledIT_ALRA
	ALRBE	LL_RTC_ALMB_Disable
		LL_RTC_ALMB_Enable
	ALRBIE	LL_RTC_DisableIT_ALRB
		LL_RTC_EnableIT_ALRB
		LL_RTC_IsEnabledIT_ALRB
	BKP	LL_RTC_TIME_DisableDayLightStore
		LL_RTC_TIME_EnableDayLightStore
		LL_RTC_TIME_IsDayLightStoreEnabled
	BYPSHAD	LL_RTC_DisableShadowRegBypass
		LL_RTC_EnableShadowRegBypass
		LL_RTC_IsShadowRegBypassEnabled

Register	Field	Function
	COE	*LL_RTC_CAL_GetOutputFreq*
		LL_RTC_CAL_SetOutputFreq
	COSEL	*LL_RTC_CAL_GetOutputFreq*
		LL_RTC_CAL_SetOutputFreq
	FMT	*LL_RTC_GetHourFormat*
		LL_RTC_SetHourFormat
	ITSE	*LL_RTC_TS_DisableInternalEvent*
		LL_RTC_TS_EnableInternalEvent
	OSEL	*LL_RTC_GetAlarmOutEvent*
		LL_RTC_SetAlarmOutEvent
	POL	*LL_RTC_GetOutputPolarity*
		LL_RTC_SetOutputPolarity
	REFCKON	*LL_RTC_DisableRefClock*
		LL_RTC_EnableRefClock
	SUB1H	*LL_RTC_TIME_DecHour*
	TSE	*LL_RTC_TS_Disable*
		LL_RTC_TS_Enable
	TSEDGE	*LL_RTC_TS_GetActiveEdge*
		LL_RTC_TS_SetActiveEdge
	TSIE	*LL_RTC_DisableIT_TS*
		LL_RTC_EnableIT_TS
		LL_RTC_IsEnabledIT_TS
	WUCKSEL	*LL_RTC_WAKEUP_GetClock*
		LL_RTC_WAKEUP_SetClock
	WUTE	*LL_RTC_WAKEUP_Disable*
		LL_RTC_WAKEUP_Enable
		LL_RTC_WAKEUP_IsEnabled
	WUTIE	*LL_RTC_DisableIT_WUT*
		LL_RTC_EnableIT_WUT
		LL_RTC_IsEnabledIT_WUT
DR	DT	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	DU	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get

Register	Field	Function
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	MT	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
	MU	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
	WDU	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetWeekDay
		LL_RTC_DATE_SetWeekDay
	YT	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetYear
		LL_RTC_DATE_SetYear
	YU	*LL_RTC_DATE_Config*
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetYear
		LL_RTC_DATE_SetYear
ISR	ALRAF	*LL_RTC_ClearFlag_ALRA*
		LL_RTC_IsActiveFlag_ALRA
	ALRAWF	*LL_RTC_IsActiveFlag_ALRAW*
	ALRBF	*LL_RTC_ClearFlag_ALRB*
		LL_RTC_IsActiveFlag_ALRB
	ALRBWF	*LL_RTC_IsActiveFlag_ALRBW*
	INIT	*LL_RTC_DisableInitMode*
		LL_RTC_EnableInitMode
	INITF	*LL_RTC_IsActiveFlag_INIT*
	INITS	*LL_RTC_IsActiveFlag_INITS*
	ITSF	*LL_RTC_ClearFlag_ITS*
		LL_RTC_IsActiveFlag_ITS
	RECALPF	*LL_RTC_IsActiveFlag_RECALP*
	RSF	*LL_RTC_ClearFlag_RS*

Register	Field	Function
		LL_RTC_IsActiveFlag_RS
	SHPF	*LL_RTC_IsActiveFlag_SHP*
	TAMP1F	*LL_RTC_ClearFlag_TAMP1*
		LL_RTC_IsActiveFlag_TAMP1
	TAMP2F	*LL_RTC_ClearFlag_TAMP2*
		LL_RTC_IsActiveFlag_TAMP2
	TAMP3F	*LL_RTC_ClearFlag_TAMP3*
		LL_RTC_IsActiveFlag_TAMP3
	TSF	*LL_RTC_ClearFlag_TS*
		LL_RTC_IsActiveFlag_TS
	TSOVF	*LL_RTC_ClearFlag_TSOV*
		LL_RTC_IsActiveFlag_TSOV
	WUTF	*LL_RTC_ClearFlag_WUT*
		LL_RTC_IsActiveFlag_WUT
	WUTWF	*LL_RTC_IsActiveFlag_WUTW*
OR	ALARMOUTTYPE	*LL_RTC_GetAlarmOutputType*
		LL_RTC_SetAlarmOutputType
	OUT_RMP	*LL_RTC_DisableOutRemap*
		LL_RTC_EnableOutRemap
PRER	PREDIV_A	*LL_RTC_GetAsynchPrescaler*
		LL_RTC_SetAsynchPrescaler
	PREDIV_S	*LL_RTC_GetSynchPrescaler*
		LL_RTC_SetSynchPrescaler
SHIFTR	ADD1S	*LL_RTC_TIME_Synchronize*
	SUBFS	*LL_RTC_TIME_Synchronize*
SSR	SS	*LL_RTC_TIME_GetSubSecond*
TAMPCR	TAMP1E	*LL_RTC_TAMPER_Disable*
		LL_RTC_TAMPER_Enable
	TAMP1IE	*LL_RTC_DisableIT_TAMP1*
		LL_RTC_EnableIT_TAMP1
		LL_RTC_IsEnabledIT_TAMP1
	TAMP1MF	*LL_RTC_TAMPER_DisableMask*
		LL_RTC_TAMPER_EnableMask
	TAMP1NOERASE	*LL_RTC_TAMPER_DisableEraseBKP*
		LL_RTC_TAMPER_EnableEraseBKP
	TAMP1TRG	*LL_RTC_TAMPER_DisableActiveLevel*

Register	Field	Function
		LL_RTC_TAMPER_EnableActiveLevel
	TAMP2E	LL_RTC_TAMPER_Disable
		LL_RTC_TAMPER_Enable
	TAMP2IE	LL_RTC_DisableIT_TAMP2
		LL_RTC_EnableIT_TAMP2
		LL_RTC_IsEnabledIT_TAMP2
	TAMP2MF	LL_RTC_TAMPER_DisableMask
		LL_RTC_TAMPER_EnableMask
	TAMP2NOERASE	LL_RTC_TAMPER_DisableEraseBKP
		LL_RTC_TAMPER_EnableEraseBKP
	TAMP2TRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
	TAMP3E	LL_RTC_TAMPER_Disable
		LL_RTC_TAMPER_Enable
	TAMP3IE	LL_RTC_DisableIT_TAMP3
		LL_RTC_EnableIT_TAMP3
		LL_RTC_IsEnabledIT_TAMP3
	TAMP3MF	LL_RTC_TAMPER_DisableMask
		LL_RTC_TAMPER_EnableMask
	TAMP3NOERASE	LL_RTC_TAMPER_DisableEraseBKP
		LL_RTC_TAMPER_EnableEraseBKP
	TAMP3TRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
	TAMPFLT	LL_RTC_TAMPER_GetFilterCount
		LL_RTC_TAMPER_SetFilterCount
	TAMPFREQ	LL_RTC_TAMPER_GetSamplingFreq
		LL_RTC_TAMPER_SetSamplingFreq
	TAMPIE	LL_RTC_DisableIT_TAMP
		LL_RTC_EnableIT_TAMP
		LL_RTC_IsEnabledIT_TAMP
	TAMPPRCH	LL_RTC_TAMPER_GetPrecharge
		LL_RTC_TAMPER_SetPrecharge
	TAMPPUDIS	LL_RTC_TAMPER_DisablePullUp
		LL_RTC_TAMPER_EnablePullUp
	TAMPTS	LL_RTC_TS_DisableOnTamper
		LL_RTC_TS_EnableOnTamper

Register	Field	Function
TR	HT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	HU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	MNT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	MNU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	PM	LL_RTC_TIME_Config
		LL_RTC_TIME_GetFormat
		LL_RTC_TIME_SetFormat
	ST	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetSecond
		LL_RTC_TIME_SetSecond
	SU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetSecond
		LL_RTC_TIME_SetSecond
TSDR	DT	LL_RTC_TS_GetDate
		LL_RTC_TS_GetDay
	DU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetDay
	MT	LL_RTC_TS_GetDate
		LL_RTC_TS_GetMonth
	MU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetMonth
	WDU	LL_RTC_TS_GetDate

Register	Field	Function
		LL_RTC_TS_GetWeekDay
TSSSR	SS	*LL_RTC_TS_GetSubSecond*
TSTR	HT	*LL_RTC_TS_GetHour*
		LL_RTC_TS_GetTime
	HU	*LL_RTC_TS_GetHour*
		LL_RTC_TS_GetTime
	MNT	*LL_RTC_TS_GetMinute*
		LL_RTC_TS_GetTime
	MNU	*LL_RTC_TS_GetMinute*
		LL_RTC_TS_GetTime
	PM	*LL_RTC_TS_GetTimeFormat*
	ST	*LL_RTC_TS_GetSecond*
		LL_RTC_TS_GetTime
	SU	*LL_RTC_TS_GetSecond*
		LL_RTC_TS_GetTime
WPR	KEY	*LL_RTC_DisableWriteProtection*
		LL_RTC_EnableWriteProtection
WUTR	WUT	*LL_RTC_WAKEUP_GetAutoReload*
		LL_RTC_WAKEUP_SetAutoReload

100.22 SPI

Table 47: Correspondence between SPI registers and SPI low-layer driver functions

Register	Field	Function
CR1	BIDIMODE	*LL_SPI_GetTransferDirection*
		LL_SPI_SetTransferDirection
	BIDIOE	*LL_SPI_GetTransferDirection*
		LL_SPI_SetTransferDirection
	BR	*LL_SPI_GetBaudRatePrescaler*
		LL_SPI_SetBaudRatePrescaler
	CPHA	*LL_SPI_GetClockPhase*
		LL_SPI_SetClockPhase
	CPOL	*LL_SPI_GetClockPolarity*
		LL_SPI_SetClockPolarity
	CRCEN	*LL_SPI_DisableCRC*
		LL_SPI_EnableCRC
		LL_SPI_IsEnabledCRC

Register	Field	Function
	CRCL	LL_SPI_GetCRCWidth
		LL_SPI_SetCRCWidth
	CRCNEXT	LL_SPI_SetCRCNext
	LSBFIRST	LL_SPI_GetTransferBitOrder
		LL_SPI_SetTransferBitOrder
	MSTR	LL_SPI_GetMode
		LL_SPI_SetMode
	RXONLY	LL_SPI_GetTransferDirection
		LL_SPI_SetTransferDirection
	SPE	LL_SPI_Disable
		LL_SPI_Enable
		LL_SPI_IsEnabled
	SSI	LL_SPI_GetMode
		LL_SPI_SetMode
	SSM	LL_SPI_GetNSSMode
		LL_SPI_SetNSSMode
CR2	DS	LL_SPI_GetDataWidth
		LL_SPI_SetDataWidth
	ERRIE	LL_SPI_DisableIT_ERR
		LL_SPI_EnableIT_ERR
		LL_SPI_IsEnabledIT_ERR
	FRF	LL_SPI_GetStandard
		LL_SPI_SetStandard
	FRXTH	LL_SPI_GetRxFIFOThreshold
		LL_SPI_SetRxFIFOThreshold
	LDMARX	LL_SPI_GetDMAParity_RX
		LL_SPI_SetDMAParity_RX
	LDMATX	LL_SPI_GetDMAParity_TX
		LL_SPI_SetDMAParity_TX
	NSSP	LL_SPI_DisableNSSPulseMgt
		LL_SPI_EnableNSSPulseMgt
		LL_SPI_IsEnabledNSSPulse
	RXDMAEN	LL_SPI_DisableDMAReq_RX
		LL_SPI_EnableDMAReq_RX
		LL_SPI_IsEnabledDMAReq_RX
	RXNEIE	LL_SPI_DisableIT_RXNE

Register	Field	Function
		LL_SPI_EnableIT_RXNE
		LL_SPI_IsEnabledIT_RXNE
	SSOE	*LL_SPI_GetNSSMode*
		LL_SPI_SetNSSMode
	TXDMAEN	*LL_SPI_DisableDMAReq_TX*
		LL_SPI_EnableDMAReq_TX
		LL_SPI_IsEnabledDMAReq_TX
	TXEIE	*LL_SPI_DisableIT_TXE*
		LL_SPI_EnableIT_TXE
		LL_SPI_IsEnabledIT_TXE
CRCPR	CRCPOLY	*LL_SPI_GetCRCPolynomial*
		LL_SPI_SetCRCPolynomial
DR	DR	*LL_SPI_DMA_GetRegAddr*
		LL_SPI_ReceiveData16
		LL_SPI_ReceiveData8
		LL_SPI_TransmitData16
		LL_SPI_TransmitData8
RXCRCR	RXCRC	*LL_SPI_GetRxCRC*
SR	BSY	*LL_SPI_IsActiveFlag_BSY*
	CRCERR	*LL_SPI_ClearFlag_CRCERR*
		LL_SPI_IsActiveFlag_CRCERR
	FRE	*LL_SPI_ClearFlag_FRE*
		LL_SPI_IsActiveFlag_FRE
	FRLVL	*LL_SPI_GetRxFIFOLevel*
	FTLVL	*LL_SPI_GetTxFIFOLevel*
	MODF	*LL_SPI_ClearFlag_MODF*
		LL_SPI_IsActiveFlag_MODF
	OVR	*LL_SPI_ClearFlag_OVR*
		LL_SPI_IsActiveFlag_OVR
	RXNE	*LL_SPI_IsActiveFlag_RXNE*
	TXE	*LL_SPI_IsActiveFlag_TXE*
TXCRCR	TXCRC	*LL_SPI_GetTxCRC*

100.23 SYSTEM

Table 48: Correspondence between SYSTEM registers and SYSTEM low-layer driver functions

Register	Field	Function

Register	Field	Function
DBGMCU_APB1FZR1	DBG_xxxx_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
DBGMCU_APB1FZR2	DBG_xxxx_STOP	*LL_DBGMCU_APB1_GRP2_FreezePeriph*
		LL_DBGMCU_APB1_GRP2_UnFreezePeriph
DBGMCU_APB2FZ	DBG_TIMx_STOP	*LL_DBGMCU_APB2_GRP1_FreezePeriph*
		LL_DBGMCU_APB2_GRP1_UnFreezePeriph
DBGMCU_CR	DBG_SLEEP	*LL_DBGMCU_DisableDBGSleepMode*
		LL_DBGMCU_EnableDBGSleepMode
	DBG_STANDBY	*LL_DBGMCU_DisableDBGStandbyMode*
		LL_DBGMCU_EnableDBGStandbyMode
	DBG_STOP	*LL_DBGMCU_DisableDBGStopMode*
		LL_DBGMCU_EnableDBGStopMode
	TRACE_IOEN	*LL_DBGMCU_GetTracePinAssignment*
		LL_DBGMCU_SetTracePinAssignment
	TRACE_MODE	*LL_DBGMCU_GetTracePinAssignment*
		LL_DBGMCU_SetTracePinAssignment
DBGMCU_IDCODE	DEV_ID	*LL_DBGMCU_GetDeviceID*
	REV_ID	*LL_DBGMCU_GetRevisionID*
FLASH_ACR	DCEN	*LL_FLASH_DisableDataCache*
		LL_FLASH_EnableDataCache
	DCRST	*LL_FLASH_DisableDataCacheReset*
		LL_FLASH_EnableDataCacheReset
	ICEN	*LL_FLASH_DisableInstCache*
		LL_FLASH_EnableInstCache
	ICRST	*LL_FLASH_DisableInstCacheReset*
		LL_FLASH_EnableInstCacheReset
	LATENCY	*LL_FLASH_GetLatency*
		LL_FLASH_SetLatency
	PRFTEN	*LL_FLASH_DisablePrefetch*
		LL_FLASH_EnablePrefetch
		LL_FLASH_IsPrefetchEnabled
	RUN_PD	*LL_FLASH_DisableRunPowerDown*
		LL_FLASH_EnableRunPowerDown
	SLEEP_PD	*LL_FLASH_DisableSleepPowerDown*
		LL_FLASH_EnableSleepPowerDown
FLASH_PDKEYR	PDKEY1	*LL_FLASH_DisableRunPowerDown*

Register	Field	Function
		LL_FLASH_EnableRunPowerDown
	PDKEY2	LL_FLASH_DisableRunPowerDown
		LL_FLASH_EnableRunPowerDown
SYSCFG_CFGR1	BOOSTEN	LL_SYSCFG_DisableAnalogBooster
		LL_SYSCFG_EnableAnalogBooster
	FPU_IE_0	LL_SYSCFG_DisableIT_FPU_IOC
		LL_SYSCFG_EnableIT_FPU_IOC
		LL_SYSCFG_IsEnabledIT_FPU_IOC
	FPU_IE_1	LL_SYSCFG_DisableIT_FPU_DZC
		LL_SYSCFG_EnableIT_FPU_DZC
		LL_SYSCFG_IsEnabledIT_FPU_DZC
	FPU_IE_2	LL_SYSCFG_DisableIT_FPU_UFC
		LL_SYSCFG_EnableIT_FPU_UFC
		LL_SYSCFG_IsEnabledIT_FPU_UFC
	FPU_IE_3	LL_SYSCFG_DisableIT_FPU_OFC
		LL_SYSCFG_EnableIT_FPU_OFC
		LL_SYSCFG_IsEnabledIT_FPU_OFC
	FPU_IE_4	LL_SYSCFG_DisableIT_FPU_IDC
		LL_SYSCFG_EnableIT_FPU_IDC
		LL_SYSCFG_IsEnabledIT_FPU_IDC
	FPU_IE_5	LL_SYSCFG_DisableIT_FPU_IXC
		LL_SYSCFG_EnableIT_FPU_IXC
		LL_SYSCFG_IsEnabledIT_FPU_IXC
	FWDIS	LL_SYSCFG_EnableFirewall
		LL_SYSCFG_IsEnabledFirewall
	I2C_PBx_FMP	LL_SYSCFG_DisableFastModePlus
		LL_SYSCFG_EnableFastModePlus
	I2Cx_FMP	LL_SYSCFG_DisableFastModePlus
		LL_SYSCFG_EnableFastModePlus
SYSCFG_CFGR2	CLL	LL_SYSCFG_GetTIMBreakInputs
		LL_SYSCFG_SetTIMBreakInputs
	ECCL	LL_SYSCFG_GetTIMBreakInputs
		LL_SYSCFG_SetTIMBreakInputs
	PVDL	LL_SYSCFG_GetTIMBreakInputs
		LL_SYSCFG_SetTIMBreakInputs
	SPF	LL_SYSCFG_ClearFlag_SP

Register	Field	Function
		LL_SYSCFG_IsActiveFlag_SP
	SPL	LL_SYSCFG_GetTIMBreakInputs
		LL_SYSCFG_SetTIMBreakInputs
SYSCFG_EXTICR1	EXTIx	LL_SYSCFG_GetEXTISource
		LL_SYSCFG_SetEXTISource
SYSCFG_EXTICR2	EXTIx	LL_SYSCFG_GetEXTISource
		LL_SYSCFG_SetEXTISource
SYSCFG_EXTICR3	EXTIx	LL_SYSCFG_GetEXTISource
		LL_SYSCFG_SetEXTISource
SYSCFG_EXTICR4	EXTIx	LL_SYSCFG_GetEXTISource
		LL_SYSCFG_SetEXTISource
SYSCFG_MEMRMP	FB_MODE	LL_SYSCFG_GetFlashBankMode
		LL_SYSCFG_SetFlashBankMode
	MEM_MODE	LL_SYSCFG_GetRemapMemory
		LL_SYSCFG_SetRemapMemory
SYSCFG_SCSR	SRAM2BSY	LL_SYSCFG_IsSRAM2EraseOngoing
	SRAM2ER	LL_SYSCFG_EnableSRAM2Erase
SYSCFG_SKR	KEY	LL_SYSCFG_LockSRAM2WRP
		LL_SYSCFG_UnlockSRAM2WRP
SYSCFG_SWPR	PxWP	LL_SYSCFG_EnableSRAM2PageWRP_0_31
SYSCFG_SWPR2	PxWP	LL_SYSCFG_EnableSRAM2PageWRP_32_63
VREFBUF_CCR	TRIM	LL_VREFBUF_GetTrimming
		LL_VREFBUF_SetTrimming
VREFBUF_CSR	ENVR	LL_VREFBUF_Disable
		LL_VREFBUF_Enable
	HIZ	LL_VREFBUF_DisableHIZ
		LL_VREFBUF_EnableHIZ
	VRR	LL_VREFBUF_IsVREFReady
	VRS	LL_VREFBUF_GetVoltageScaling
		LL_VREFBUF_SetVoltageScaling

100.24 TIM

Table 49: Correspondence between TIM registers and TIM low-layer driver functions

Register	Field	Function
ARR	ARR	LL_TIM_GetAutoReload
		LL_TIM_SetAutoReload

Register	Field	Function
BDTR	AOE	LL_TIM_DisableAutomaticOutput
		LL_TIM_EnableAutomaticOutput
		LL_TIM_IsEnabledAutomaticOutput
	BK2E	LL_TIM_DisableBRK2
		LL_TIM_EnableBRK2
	BK2F	LL_TIM_ConfigBRK2
	BK2P	LL_TIM_ConfigBRK2
	BKE	LL_TIM_DisableBRK
		LL_TIM_EnableBRK
	BKF	LL_TIM_ConfigBRK
	BKP	LL_TIM_ConfigBRK
	DTG	LL_TIM_OC_SetDeadTime
	LOCK	LL_TIM_CC_SetLockLevel
	MOE	LL_TIM_DisableAllOutputs
		LL_TIM_EnableAllOutputs
		LL_TIM_IsEnabledAllOutputs
	OSSI	LL_TIM_SetOffStates
	OSSR	LL_TIM_SetOffStates
CCER	CC1E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC1NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC1NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC1P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC2E	LL_TIM_CC_DisableChannel

Register	Field	Function
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC2NE	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC2NP	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC2P	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3E	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NE	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NP	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3P	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC4E	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel

Register	Field	Function
	CC4NP	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
	CC4P	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC5E	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC5P	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC6E	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC6P	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
CCMR1	CC1S	*LL_TIM_IC_Config*
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	CC2S	*LL_TIM_IC_Config*
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC1F	*LL_TIM_IC_Config*
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC1PSC	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	IC2F	*LL_TIM_IC_Config*

Register	Field	Function
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC2PSC	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	OC1CE	*LL_TIM_OC_DisableClear*
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC1FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC1M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC1PE	*LL_TIM_OC_DisablePreload*
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC2CE	*LL_TIM_OC_DisableClear*
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC2FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC2M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC2PE	*LL_TIM_OC_DisablePreload*
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCMR2	CC3S	*LL_TIM_IC_Config*
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	CC4S	*LL_TIM_IC_Config*
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC3F	*LL_TIM_IC_Config*

Register	Field	Function
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC3PSC	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	IC4F	*LL_TIM_IC_Config*
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC4PSC	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	OC3CE	*LL_TIM_OC_DisableClear*
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC3FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC3M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC3PE	*LL_TIM_OC_DisablePreload*
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC4CE	*LL_TIM_OC_DisableClear*
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC4FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC4M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC4PE	*LL_TIM_OC_DisablePreload*
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCMR3	CC5S	*LL_TIM_OC_ConfigOutput*
	CC6S	*LL_TIM_OC_ConfigOutput*
	OC5CE	*LL_TIM_OC_DisableClear*

Register	Field	Function
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC5FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC5M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC5PE	*LL_TIM_OC_DisablePreload*
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC6CE	*LL_TIM_OC_DisableClear*
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC6FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC6M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC6PE	*LL_TIM_OC_DisablePreload*
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCR1	CCR1	*LL_TIM_IC_GetCaptureCH1*
		LL_TIM_OC_GetCompareCH1
		LL_TIM_OC_SetCompareCH1
CCR2	CCR2	*LL_TIM_IC_GetCaptureCH2*
		LL_TIM_OC_GetCompareCH2
		LL_TIM_OC_SetCompareCH2
CCR3	CCR3	*LL_TIM_IC_GetCaptureCH3*
		LL_TIM_OC_GetCompareCH3
		LL_TIM_OC_SetCompareCH3
CCR4	CCR4	*LL_TIM_IC_GetCaptureCH4*
		LL_TIM_OC_GetCompareCH4
		LL_TIM_OC_SetCompareCH4
CCR5	CCR5	*LL_TIM_OC_GetCompareCH5*
		LL_TIM_OC_SetCompareCH5
	GC5C1	*LL_TIM_SetCH5CombinedChannels*

Register	Field	Function
	GC5C2	*LL_TIM_SetCH5CombinedChannels*
	GC5C3	*LL_TIM_SetCH5CombinedChannels*
CCR6	CCR6	*LL_TIM_OC_GetCompareCH6*
		LL_TIM_OC_SetCompareCH6
CNT	CNT	*LL_TIM_GetCounter*
		LL_TIM_SetCounter
CR1	ARPE	*LL_TIM_DisableARRPreload*
		LL_TIM_EnableARRPreload
		LL_TIM_IsEnabledARRPreload
	CEN	*LL_TIM_DisableCounter*
		LL_TIM_EnableCounter
		LL_TIM_IsEnabledCounter
	CKD	*LL_TIM_GetClockDivision*
		LL_TIM_SetClockDivision
	CMS	*LL_TIM_GetCounterMode*
		LL_TIM_SetCounterMode
	DIR	*LL_TIM_GetCounterMode*
		LL_TIM_GetDirection
		LL_TIM_SetCounterMode
	OPM	*LL_TIM_GetOnePulseMode*
		LL_TIM_SetOnePulseMode
	UDIS	*LL_TIM_DisableUpdateEvent*
		LL_TIM_EnableUpdateEvent
		LL_TIM_IsEnabledUpdateEvent
	UIFREMAP	*LL_TIM_DisableUIFRemap*
		LL_TIM_EnableUIFRemap
	URS	*LL_TIM_GetUpdateSource*
		LL_TIM_SetUpdateSource
CR2	CCDS	*LL_TIM_CC_GetDMAReqTrigger*
		LL_TIM_CC_SetDMAReqTrigger
	CCPC	*LL_TIM_CC_DisablePreload*
		LL_TIM_CC_EnablePreload
	CCUS	*LL_TIM_CC_SetUpdate*
	MMS	*LL_TIM_SetTriggerOutput*
	MMS2	*LL_TIM_SetTriggerOutput2*
	OIS1	*LL_TIM_OC_ConfigOutput*

Register	Field	Function
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS2	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS2N	*LL_TIM_OC_GetIdleState*
		LL_TIM_OC_SetIdleState
	OIS3	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS3N	*LL_TIM_OC_GetIdleState*
		LL_TIM_OC_SetIdleState
	OIS4	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS5	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS6	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	TI1S	*LL_TIM_IC_DisableXORCombination*
		LL_TIM_IC_EnableXORCombination
		LL_TIM_IC_IsEnabledXORCombination
DCR	DBA	*LL_TIM_ConfigDMABurst*
	DBL	*LL_TIM_ConfigDMABurst*
DIER	BIE	*LL_TIM_DisableIT_BRK*
		LL_TIM_EnableIT_BRK
		LL_TIM_IsEnabledIT_BRK
	CC1DE	*LL_TIM_DisableDMAReq_CC1*
		LL_TIM_EnableDMAReq_CC1
		LL_TIM_IsEnabledDMAReq_CC1
	CC1IE	*LL_TIM_DisableIT_CC1*
		LL_TIM_EnableIT_CC1
		LL_TIM_IsEnabledIT_CC1
	CC2DE	*LL_TIM_DisableDMAReq_CC2*

Register	Field	Function
		LL_TIM_EnableDMAReq_CC2
		LL_TIM_IsEnabledDMAReq_CC2
	CC2IE	LL_TIM_DisableIT_CC2
		LL_TIM_EnableIT_CC2
		LL_TIM_IsEnabledIT_CC2
	CC3DE	LL_TIM_DisableDMAReq_CC3
		LL_TIM_EnableDMAReq_CC3
		LL_TIM_IsEnabledDMAReq_CC3
	CC3IE	LL_TIM_DisableIT_CC3
		LL_TIM_EnableIT_CC3
		LL_TIM_IsEnabledIT_CC3
	CC4DE	LL_TIM_DisableDMAReq_CC4
		LL_TIM_EnableDMAReq_CC4
		LL_TIM_IsEnabledDMAReq_CC4
	CC4IE	LL_TIM_DisableIT_CC4
		LL_TIM_EnableIT_CC4
		LL_TIM_IsEnabledIT_CC4
	COMDE	LL_TIM_DisableDMAReq_COM
		LL_TIM_EnableDMAReq_COM
		LL_TIM_IsEnabledDMAReq_COM
	COMIE	LL_TIM_DisableIT_COM
		LL_TIM_EnableIT_COM
		LL_TIM_IsEnabledIT_COM
	TDE	LL_TIM_DisableDMAReq_TRIG
		LL_TIM_EnableDMAReq_TRIG
		LL_TIM_IsEnabledDMAReq_TRIG
	TIE	LL_TIM_DisableIT_TRIG
		LL_TIM_EnableIT_TRIG
		LL_TIM_IsEnabledIT_TRIG
	UDE	LL_TIM_DisableDMAReq_UPDATE
		LL_TIM_EnableDMAReq_UPDATE
		LL_TIM_IsEnabledDMAReq_UPDATE
	UIE	LL_TIM_DisableIT_UPDATE
		LL_TIM_EnableIT_UPDATE
		LL_TIM_IsEnabledIT_UPDATE
EGR	B2G	LL_TIM_GenerateEvent_BRK2

Register	Field	Function
	BG	*LL_TIM_GenerateEvent_BRK*
	CC1G	*LL_TIM_GenerateEvent_CC1*
	CC2G	*LL_TIM_GenerateEvent_CC2*
	CC3G	*LL_TIM_GenerateEvent_CC3*
	CC4G	*LL_TIM_GenerateEvent_CC4*
	COMG	*LL_TIM_GenerateEvent_COM*
	TG	*LL_TIM_GenerateEvent_TRIG*
	UG	*LL_TIM_GenerateEvent_UPDATE*
OR2	BKCMP1E	*LL_TIM_DisableBreakInputSource*
		LL_TIM_EnableBreakInputSource
		LL_TIM_SetBreakInputSourcePolarity
	BKCMP2E	*LL_TIM_DisableBreakInputSource*
		LL_TIM_EnableBreakInputSource
		LL_TIM_SetBreakInputSourcePolarity
	BKDFBK0E	*LL_TIM_DisableBreakInputSource*
		LL_TIM_EnableBreakInputSource
	BKINE	*LL_TIM_DisableBreakInputSource*
		LL_TIM_EnableBreakInputSource
		LL_TIM_SetBreakInputSourcePolarity
	BKINP	*LL_TIM_SetBreakInputSourcePolarity*
	ETRSEL	*LL_TIM_SetETRSource*
OR3	BKCMP1E	*LL_TIM_DisableBreakInputSource*
		LL_TIM_EnableBreakInputSource
		LL_TIM_SetBreakInputSourcePolarity
	BKCMP2E	*LL_TIM_DisableBreakInputSource*
		LL_TIM_EnableBreakInputSource
		LL_TIM_SetBreakInputSourcePolarity
	BKDFBK0E	*LL_TIM_DisableBreakInputSource*
		LL_TIM_EnableBreakInputSource
	BKINE	*LL_TIM_DisableBreakInputSource*
		LL_TIM_EnableBreakInputSource
		LL_TIM_SetBreakInputSourcePolarity
	BKINP	*LL_TIM_SetBreakInputSourcePolarity*
PSC	PSC	*LL_TIM_GetPrescaler*
		LL_TIM_SetPrescaler
RCR	REP	*LL_TIM_GetRepetitionCounter*

Register	Field	Function
		LL_TIM_SetRepetitionCounter
SMCR	ECE	*LL_TIM_DisableExternalClock*
		LL_TIM_EnableExternalClock
		LL_TIM_IsEnabledExternalClock
		LL_TIM_SetClockSource
	ETF	*LL_TIM_ConfigETR*
	ETP	*LL_TIM_ConfigETR*
	ETPS	*LL_TIM_ConfigETR*
	MSM	*LL_TIM_DisableMasterSlaveMode*
		LL_TIM_EnableMasterSlaveMode
		LL_TIM_IsEnabledMasterSlaveMode
	OCCS	*LL_TIM_SetOCRefClearInputSource*
	SMS	*LL_TIM_SetClockSource*
		LL_TIM_SetEncoderMode
		LL_TIM_SetSlaveMode
	TS	*LL_TIM_SetTriggerInput*
SR	B2IF	*LL_TIM_ClearFlag_BRK2*
		LL_TIM_IsActiveFlag_BRK2
	BIF	*LL_TIM_ClearFlag_BRK*
		LL_TIM_IsActiveFlag_BRK
	CC1IF	*LL_TIM_ClearFlag_CC1*
		LL_TIM_IsActiveFlag_CC1
	CC1OF	*LL_TIM_ClearFlag_CC1OVR*
		LL_TIM_IsActiveFlag_CC1OVR
	CC2IF	*LL_TIM_ClearFlag_CC2*
		LL_TIM_IsActiveFlag_CC2
	CC2OF	*LL_TIM_ClearFlag_CC2OVR*
		LL_TIM_IsActiveFlag_CC2OVR
	CC3IF	*LL_TIM_ClearFlag_CC3*
		LL_TIM_IsActiveFlag_CC3
	CC3OF	*LL_TIM_ClearFlag_CC3OVR*
		LL_TIM_IsActiveFlag_CC3OVR
	CC4IF	*LL_TIM_ClearFlag_CC4*
		LL_TIM_IsActiveFlag_CC4
	CC4OF	*LL_TIM_ClearFlag_CC4OVR*
		LL_TIM_IsActiveFlag_CC4OVR

Register	Field	Function
	CC5IF	*LL_TIM_ClearFlag_CC5*
		LL_TIM_IsActiveFlag_CC5
	CC6IF	*LL_TIM_ClearFlag_CC6*
		LL_TIM_IsActiveFlag_CC6
	COMIF	*LL_TIM_ClearFlag_COM*
		LL_TIM_IsActiveFlag_COM
	SBIF	*LL_TIM_ClearFlag_SYSBRK*
		LL_TIM_IsActiveFlag_SYSBRK
	TIF	*LL_TIM_ClearFlag_TRIG*
		LL_TIM_IsActiveFlag_TRIG
	UIF	*LL_TIM_ClearFlag_UPDATE*
		LL_TIM_IsActiveFlag_UPDATE

100.25 USART

Table 50: Correspondence between USART registers and USART low-layer driver functions

Register	Field	Function
BRR	BRR	*LL_USART_GetBaudRate*
		LL_USART_SetBaudRate
CR1	CMIE	*LL_USART_DisableIT_CM*
		LL_USART_EnableIT_CM
		LL_USART_IsEnabledIT_CM
	DEAT	*LL_USART_GetDEAssertionTime*
		LL_USART_SetDEAssertionTime
	DEDT	*LL_USART_GetDEDeassertionTime*
		LL_USART_SetDEDeassertionTime
	EOBIE	*LL_USART_DisableIT_EOB*
		LL_USART_EnableIT_EOB
		LL_USART_IsEnabledIT_EOB
	FIFOEN	*LL_USART_DisableFIFO*
		LL_USART_EnableFIFO
		LL_USART_IsEnabledFIFO
	IDLEIE	*LL_USART_DisableIT_IDLE*
		LL_USART_EnableIT_IDLE
		LL_USART_IsEnabledIT_IDLE
	M0	*LL_USART_ConfigCharacter*
		LL_USART_GetDataWidth

Register	Field	Function
		LL_USART_SetDataWidth
	M1	*LL_USART_ConfigCharacter*
		LL_USART_GetDataWidth
		LL_USART_SetDataWidth
	MME	*LL_USART_DisableMuteMode*
		LL_USART_EnableMuteMode
		LL_USART_IsEnabledMuteMode
	OVER8	*LL_USART_GetOverSampling*
		LL_USART_SetOverSampling
	PCE	*LL_USART_ConfigCharacter*
		LL_USART_GetParity
		LL_USART_SetParity
	PEIE	*LL_USART_DisableIT_PE*
		LL_USART_EnableIT_PE
		LL_USART_IsEnabledIT_PE
	PS	*LL_USART_ConfigCharacter*
		LL_USART_GetParity
		LL_USART_SetParity
	RE	*LL_USART_DisableDirectionRx*
		LL_USART_EnableDirectionRx
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
	RTOIE	*LL_USART_DisableIT_RTO*
		LL_USART_EnableIT_RTO
		LL_USART_IsEnabledIT_RTO
	RXFFIE	*LL_USART_DisableIT_RXFF*
		LL_USART_EnableIT_RXFF
		LL_USART_IsEnabledIT_RXFF
	RXNEIE_RXFNEIE	*LL_USART_DisableIT_RXNE_RXFNE*
		LL_USART_EnableIT_RXNE_RXFNE
		LL_USART_IsEnabledIT_RXNE_RXFNE
	TCIE	*LL_USART_DisableIT_TC*
		LL_USART_EnableIT_TC
		LL_USART_IsEnabledIT_TC
	TE	*LL_USART_DisableDirectionTx*
		LL_USART_EnableDirectionTx

Register	Field	Function
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
	TXEIE_TXFNFIE	LL_USART_DisableIT_TXE_TXFNF
		LL_USART_EnableIT_TXE_TXFNF
		LL_USART_IsEnabledIT_TXE_TXFNF
	TXFEIE	LL_USART_DisableIT_TXFE
		LL_USART_EnableIT_TXFE
		LL_USART_IsEnabledIT_TXFE
	UE	LL_USART_Disable
		LL_USART_Enable
		LL_USART_IsEnabled
	UESM	LL_USART_DisableInStopMode
		LL_USART_EnableInStopMode
		LL_USART_IsEnabledInStopMode
	WAKE	LL_USART_GetWakeUpMethod
		LL_USART_SetWakeUpMethod
CR2	ABREN	LL_USART_DisableAutoBaudRate
		LL_USART_EnableAutoBaudRate
		LL_USART_IsEnabledAutoBaud
	ABRMODE	LL_USART_GetAutoBaudRateMode
		LL_USART_SetAutoBaudRateMode
	ADD	LL_USART_ConfigNodeAddress
		LL_USART_GetNodeAddress
	ADDM7	LL_USART_ConfigNodeAddress
		LL_USART_GetNodeAddressLen
	CLKEN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableSCLKOutput
		LL_USART_EnableSCLKOutput
		LL_USART_IsEnabledSCLKOutput
	CPHA	LL_USART_ConfigClock

Register	Field	Function
		LL_USART_GetClockPhase
		LL_USART_SetClockPhase
	CPOL	*LL_USART_ConfigClock*
		LL_USART_GetClockPolarity
		LL_USART_SetClockPolarity
	DATAINV	*LL_USART_GetBinaryDataLogic*
		LL_USART_SetBinaryDataLogic
	DIS_NSS	*LL_USART_DisableSPISlaveSelect*
		LL_USART_EnableSPISlaveSelect
		LL_USART_IsEnabledSPISlaveSelect
	LBCL	*LL_USART_ConfigClock*
		LL_USART_GetLastClkPulseOutput
		LL_USART_SetLastClkPulseOutput
	LBDIE	*LL_USART_DisableIT_LBD*
		LL_USART_EnableIT_LBD
		LL_USART_IsEnabledIT_LBD
	LBDL	*LL_USART_GetLINBrkDetectionLen*
		LL_USART_SetLINBrkDetectionLen
	LINEN	*LL_USART_ConfigAsyncMode*
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableLIN
		LL_USART_EnableLIN
		LL_USART_IsEnabledLIN
	MSBFIRST	*LL_USART_GetTransferBitOrder*
		LL_USART_SetTransferBitOrder
	RTOEN	*LL_USART_DisableRxTimeout*
		LL_USART_EnableRxTimeout
		LL_USART_IsEnabledRxTimeout
	RXINV	*LL_USART_GetRXPinLevel*
		LL_USART_SetRXPinLevel
	SLVEN	*LL_USART_DisableSPISlave*

Register	Field	Function
		LL_USART_EnableSPISlave
		LL_USART_IsEnabledSPISlave
	STOP	*LL_USART_ConfigCharacter*
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigSmartcardMode
		LL_USART_GetStopBitsLength
		LL_USART_SetStopBitsLength
	SWAP	*LL_USART_GetTXRXSwap*
		LL_USART_SetTXRXSwap
	TXINV	*LL_USART_GetTXPinLevel*
		LL_USART_SetTXPinLevel
CR3	CTSE	*LL_USART_DisableCTSHWFlowCtrl*
		LL_USART_EnableCTSHWFlowCtrl
		LL_USART_GetHWFlowCtrl
		LL_USART_SetHWFlowCtrl
	CTSIE	*LL_USART_DisableIT_CTS*
		LL_USART_EnableIT_CTS
		LL_USART_IsEnabledIT_CTS
	DDRE	*LL_USART_DisableDMADeactOnRxErr*
		LL_USART_EnableDMADeactOnRxErr
		LL_USART_IsEnabledDMADeactOnRxErr
	DEM	*LL_USART_DisableDEMode*
		LL_USART_EnableDEMode
		LL_USART_IsEnabledDEMode
	DEP	*LL_USART_GetDESignalPolarity*
		LL_USART_SetDESignalPolarity
	DMAR	*LL_USART_DisableDMAReq_RX*
		LL_USART_EnableDMAReq_RX
		LL_USART_IsEnabledDMAReq_RX
	DMAT	*LL_USART_DisableDMAReq_TX*
		LL_USART_EnableDMAReq_TX
		LL_USART_IsEnabledDMAReq_TX
	EIE	*LL_USART_DisableIT_ERROR*
		LL_USART_EnableIT_ERROR
		LL_USART_IsEnabledIT_ERROR

Register	Field	Function
	HDSEL	*LL_USART_ConfigAsyncMode*
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableHalfDuplex
		LL_USART_EnableHalfDuplex
		LL_USART_IsEnabledHalfDuplex
	IREN	*LL_USART_ConfigAsyncMode*
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableIrda
		LL_USART_EnableIrda
		LL_USART_IsEnabledIrda
	IRLP	*LL_USART_GetIrdaPowerMode*
		LL_USART_SetIrdaPowerMode
	NACK	*LL_USART_DisableSmartcardNACK*
		LL_USART_EnableSmartcardNACK
		LL_USART_IsEnabledSmartcardNACK
	ONEBIT	*LL_USART_DisableOneBitSamp*
		LL_USART_EnableOneBitSamp
		LL_USART_IsEnabledOneBitSamp
	OVRDIS	*LL_USART_DisableOverrunDetect*
		LL_USART_EnableOverrunDetect
		LL_USART_IsEnabledOverrunDetect
	RTSE	*LL_USART_DisableRTSHWFlowCtrl*
		LL_USART_EnableRTSHWFlowCtrl
		LL_USART_GetHWFlowCtrl
		LL_USART_SetHWFlowCtrl
	RXFTCFG	*LL_USART_ConfigFIFOsThreshold*
		LL_USART_GetRXFIFOThreshold

Register	Field	Function
		LL_USART_SetRXFIFOThreshold
	RXFTIE	*LL_USART_DisableIT_RXFT*
		LL_USART_EnableIT_RXFT
		LL_USART_IsEnabledIT_RXFT
	SCARCNT	*LL_USART_GetSmartcardAutoRetryCount*
		LL_USART_SetSmartcardAutoRetryCount
	SCEN	*LL_USART_ConfigAsyncMode*
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableSmartcard
		LL_USART_EnableSmartcard
		LL_USART_IsEnabledSmartcard
	TCBGTIE	*LL_USART_DisableIT_TCBGT*
		LL_USART_EnableIT_TCBGT
		LL_USART_IsEnabledIT_TCBGT
	TXFTCFG	*LL_USART_ConfigFIFOsThreshold*
		LL_USART_GetTXFIFOThreshold
		LL_USART_SetTXFIFOThreshold
	TXFTIE	*LL_USART_DisableIT_TXFT*
		LL_USART_EnableIT_TXFT
		LL_USART_IsEnabledIT_TXFT
	WUFIE	*LL_USART_DisableIT_WKUP*
		LL_USART_EnableIT_WKUP
		LL_USART_IsEnabledIT_WKUP
	WUS	*LL_USART_GetWKUPType*
		LL_USART_SetWKUPType
GTPR	GT	*LL_USART_GetSmartcardGuardTime*
		LL_USART_SetSmartcardGuardTime
	PSC	*LL_USART_GetIrdaPrescaler*
		LL_USART_GetSmartcardPrescaler
		LL_USART_SetIrdaPrescaler
		LL_USART_SetSmartcardPrescaler

Register	Field	Function
ICR	CMCF	*LL_USART_ClearFlag_CM*
	CTSCF	*LL_USART_ClearFlag_nCTS*
	EOBCF	*LL_USART_ClearFlag_EOB*
	FECF	*LL_USART_ClearFlag_FE*
	IDLECF	*LL_USART_ClearFlag_IDLE*
	LBDCF	*LL_USART_ClearFlag_LBD*
	NCF	*LL_USART_ClearFlag_NE*
	ORECF	*LL_USART_ClearFlag_ORE*
	PECF	*LL_USART_ClearFlag_PE*
	RTOCF	*LL_USART_ClearFlag_RTO*
	TCBGTCF	*LL_USART_ClearFlag_TCBGT*
	TCCF	*LL_USART_ClearFlag_TC*
	TXFECF	*LL_USART_ClearFlag_TXFE*
	UDRCF	*LL_USART_ClearFlag_UDR*
	WUCF	*LL_USART_ClearFlag_WKUP*
ISR	ABRE	*LL_USART_IsActiveFlag_ABRE*
	ABRF	*LL_USART_IsActiveFlag_ABR*
	BUSY	*LL_USART_IsActiveFlag_BUSY*
	CMF	*LL_USART_IsActiveFlag_CM*
	CTS	*LL_USART_IsActiveFlag_CTS*
	CTSIF	*LL_USART_IsActiveFlag_nCTS*
	EOBF	*LL_USART_IsActiveFlag_EOB*
	FE	*LL_USART_IsActiveFlag_FE*
	IDLE	*LL_USART_IsActiveFlag_IDLE*
	LBDF	*LL_USART_IsActiveFlag_LBD*
	NF	*LL_USART_IsActiveFlag_NE*
	ORE	*LL_USART_IsActiveFlag_ORE*
	PE	*LL_USART_IsActiveFlag_PE*
	REACK	*LL_USART_IsActiveFlag_REACK*
	RTOF	*LL_USART_IsActiveFlag_RTO*
	RWU	*LL_USART_IsActiveFlag_RWU*
	RXFF	*LL_USART_IsActiveFlag_RXFF*
	RXFT	*LL_USART_IsActiveFlag_RXFT*
	RXNE_RXFNE	*LL_USART_IsActiveFlag_RXNE_RXFNE*
	SBKF	*LL_USART_IsActiveFlag_SBK*
	TC	*LL_USART_IsActiveFlag_TC*

Register	Field	Function
	TCBGT	LL_USART_IsActiveFlag_TCBGT
	TEACK	LL_USART_IsActiveFlag_TEACK
	TXE_TXFNF	LL_USART_IsActiveFlag_TXE_TXFNF
	TXFE	LL_USART_IsActiveFlag_TXFE
	TXFT	LL_USART_IsActiveFlag_TXFT
	UDR	LL_USART_IsActiveFlag_UDR
	WUF	LL_USART_IsActiveFlag_WKUP
PRESC	PRESCALER	LL_USART_GetPrescaler
		LL_USART_SetPrescaler
RDR	RDR	LL_USART_DMA_GetRegAddr
		LL_USART_ReceiveData8
		LL_USART_ReceiveData9
RQR	ABRRQ	LL_USART_RequestAutoBaudRate
	MMRQ	LL_USART_RequestEnterMuteMode
	RXFRQ	LL_USART_RequestRxDataFlush
	SBKRQ	LL_USART_RequestBreakSending
	TXFRQ	LL_USART_RequestTxDataFlush
RTOR	BLEN	LL_USART_GetBlockLength
		LL_USART_SetBlockLength
	RTO	LL_USART_GetRxTimeout
		LL_USART_SetRxTimeout
TDR	TDR	LL_USART_DMA_GetRegAddr
		LL_USART_TransmitData8
		LL_USART_TransmitData9

100.26 WWDG

Table 51: Correspondence between WWDG registers and WWDG low-layer driver functions

Register	Field	Function
CFR	EWI	LL_WWDG_EnableIT_EWKUP
		LL_WWDG_IsEnabledIT_EWKUP
	W	LL_WWDG_GetWindow
		LL_WWDG_SetWindow
	WDGTB	LL_WWDG_GetPrescaler
		LL_WWDG_SetPrescaler
CR	T	LL_WWDG_GetCounter
		LL_WWDG_SetCounter

Register	Field	Function
	WDGA	*LL_WWDG_Enable*
		LL_WWDG_IsEnabled
SR	EWIF	*LL_WWDG_ClearFlag_EWKUP*
		LL_WWDG_IsActiveFlag_EWKUP

101 FAQs

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  – Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  – Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not require in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, …). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL_UART_Init() then HAL_UART_Transmit() or HAL_UART_Receive().

Which STM32L4 series and STM32L4+ series devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32L4 series and STM32L4+ series devices. To ensure compatibility between all devices and portability with others series and lines, the API is split into the generic and the extension APIs . For more details, please refer to *Section 2.4: "Devices supported by HAL drivers"*.

What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

Architecture

How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32l4xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration…)

A template is provided in the HAL drivers folders (stm32l4xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32l4xx_hal.h file has to be included.

What is the difference between stm32l4xx_hal_ppp.c/.h andstm32l4xx_hal_ppp_*ex*.c/.h?

The HAL driver architecture supports common features across STM32 series/lines. To support specific features, the drivers are split into two groups.

- The generic APIs (xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (xx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

Initialization and I/O operation functions

How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system_xx.c) but in the main user application by calling the two main functions, HAL_RCC_OscConfig() and HAL_RCC_ClockConfig(). It can be modified in any user application section.

What is the purpose of the *PPP_HandleTypeDef *pHandle* structure located in each driver in addition to the Initialization structure

PPP_HandleTypeDef *pHandle is the main structure implemented in the HAL drivers. It handles the peripheral configuration and registers, and embeds all the structures and variables required to follow the peripheral device flow (pointer to buffer, Error code, State,...)

However, this structure is not required to service peripherals such as GPIO, SYSTICK, PWR, and RCC.

What is the purpose of HAL_PPP_MspInit() and HAL_PPP_MspDeInit() functions?

These function are called within HAL_PPP_Init() and HAL_PPP_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in xx_hal_msp.c. A template is provided in the HAL driver folders (xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute __*weak*)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use HAL_Init() function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,…), configure the systTick and the NVIC priority grouping and the hardware low level initialization.

The SysTick configuration shall be adjusted by calling ***HAL_RCC_ClockConfig()*** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling *HAL_IncTick()* function in SysTick ISR and retrieve the value of this variable by calling *HAL_GetTick()* function.

The call HAL_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL_Delay().

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,…).
2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling HAL_PPP_MspInit() inxx_hal_msp.c
6. Start your process operation by calling IO operation functions.

What is the purpose of HAL_PPP_IRQHandler() function and when should I use it?

HAL_PPP_IRQHandler() is used to handle interrupt process. It is called under PPP_IRQHandler() function in xx_it.c. In this case, the end-user has to implement only the callbacks functions (prefixed by __weak) to perform the appropriate action when an interrupt is detected. Advanced users can implement their own code in PPP_IRQHandler() without calling HAL_PPP_IRQHandler().

Can I use directly the macros defined in xx_hal_ppp.h ?

Yes, you can: a set of macros is provided with the APIs. They allow accessing directly some specific features using peripheral flags.

Where must PPP_HandleTypedef structure peripheral handler be declared?

PPP_HandleTypedef structure peripheral handler must be declared as a global variable, so that all the structure fields are set to 0 by default. In this way, the peripheral handler default state are set to HAL_PPP_STATE_RESET, which is the default state for each peripheral after a system reset.

When should I use HAL versus LL drivers?

HAL drivers offer high-level and function-oriented APIs, with a high level of portability. Product/IPs complexity is hidden for end users. LL drivers offer low-level APIs at registers level, with a better optimization but less portability. They require a deep knowledge of product/IPs specifications.

How can I include LL drivers in my environment? Is there any LL configuration file as for HAL?

There is no configuration file. Source code shall directly include the necessary stm32l4xx_ll_ppp.h file(s).

Can I use HAL and LL drivers together? If yes, what are the constraints?

It is possible to use both HAL and LL drivers. One can handle the IP initialization phase with HAL and then manage the I/O operations with LL drivers. The major difference between HAL and LL is that HAL drivers require to create and use handles for operation management while LL drivers operates directly on peripheral registers. Mixing HAL and LL is illustrated in Examples_MIX example.

Is there any LL APIs which are not available with HAL?

Yes, there are. A few Cortex® APIs have been added in stm32l4xx_ll_cortex.h e.g. for accessing SCB or SysTick registers.

Why are SysTick interrupts not enabled on LL drivers?

When using LL drivers in standalone mode, you do not need to enable SysTick interrupts because they are not used in LL APIs, while HAL functions requires SysTick interrupts to manage timeouts.

102 Revision history

Table 52: Document revision history

Date	Revision	Changes
10-Jun-2015	1	Initial release.
07-Jul-2015	2	Document classification changed to public without content changes.
17-Sep-2015	3	Added LSI in *Table 12: "Define statements used for HAL configuration"*. Updated *Table 13: "Description of GPIO_InitTypeDef structure"* Added low-layer driver APIs. Updated STM32L4 new low-power management features in *Section 2.11.4: "PWR"*. *Section 11: "HAL CRC Extension Driver"*: Added HAL_CRYPEX_ProcessSuspend() API. *Section 24: "HAL FLASH Generic Driver"*: Added FLASH_OB_USER_nRST_STANDBY parameter value for USERConfig field used in HAL_FLASHEx_OBProgram() and HAL_FLASHEx_OBGetConfig(). *Section 24: "HAL FLASH Generic Driver"*: Added RCC_MCO1SOURCE_NOCLOCK parameter value for HAL_RCC_MCOConfig(). *Section 24: "HAL FLASH Generic Driver"*: • Added HAL_RCCEx_EnableLSECSS_IT(),HAL_RCCEx_LSECSS_IRQHandler() and HAL_RCCEx_LSECSS_Callback() APIs. • Updated HAL_RCCEx_GetPeriphCLKFreq() to support all peripherals with different clock sources.
07-Dec-2015	4	*Section 2.11.4: "PWR"*: • Renamed 'STOP1 with main regulator on' into 'STOP0'. • Restricted STOP1' to low-power regulator on. • Updated *Section 2.11.4: "PWR"* and : added STOP0 mode in function description. *Section 50: "HAL PWR Extension Driver"*: • Added function. • Updated prototype (*Regulator* parameter removed) Updated SyncExt field in *Section 56.1.2: "SAI_InitTypeDef"* structure: replaced SAI_SYNCHRONOUS_EXT value by SAI_SYNCHRONOUS_EXT_SAI1 and SAI_SYNCHRONOUS_EXT_SAI2. *Section 35.3: "IRDA Firmware driver defines"*: corrected __HAL_IRDA_GET_IT_SOURCE() description. *Section 59.3: "SMARTCARD Firmware driver defines"*: corrected __HAL_SMARTCARD_GET_IT_SOURCE() description. Updated *LowPowerMode* parameter and changed return value: • Renamed LL_PWR_MODE_STOP1_MAIN_REGU power mode into LL_PWR_MODE_STOP0. • Renamed LL_PWR_MODE_STOP1_LP_REGU power mode into LL_PWR_MODE_STOP1. Added function. Added LL_RCC_LSE_DisableCSS() in *Section 100.19: "RCC"*.

Date	Revision	Changes
19-Feb-2016	5	Updated LL driver features in *Section 3: "Overview of low-layer drivers"*. Updated *Section 3.1: "Low-layer files"*. Added low-layers driver initialization and de-initialization APIs (when applicable) for LL ADC, LL COMP, LL DAC, LL DMA, LL EXTI, LL GPIO, LL I2C, LL LPTIM, LL LPUART, LL OPAMP, LL PWR, LL RCC, LL RTC, LL SPI, LL SWPMI, LL TIM and LL USART. **HAL I2C** Generic driver: • The following new APIs now support repeated start feature: – HAL_I2C_Master_Sequential_Transmit_IT(), HAL_I2C_Master_Sequential_Receive_IT() and HAL_I2C_Master_Abort_IT(), – HAL_I2C_Slave_Sequential_Transmit_IT() and HAL_I2C_Slave_Sequential_Receive_IT() – HAL_I2C_EnableListen_IT() and HAL_I2C_DisableListen_IT() • New user callbacks HAL_I2C_ListenCpltCallback() and HAL_I2C_AddrCallback() • New API HAL_I2C_GetMode() to return HAL_I2C_MODE_MASTER, HAL_I2C_MODE_SLAVE or HAL_I2C_MODE_NONE **HAL IRDA** Generic driver: • Added missing IRDA_CLEAR_IDLEF definition for IDLE flag clear with __HAL_IRDA_CLEAR_FLAG(). **HAL SMARTCARD** Generic driver: • Added missing SMARTCARD_STOPBITS_0_5 definition for 0.5 stop bit frames. **HAL UART** Generic driver: • Added missing UART_STOPBITS_0_5 definition for 0.5 stop bit frames. **HAL USART** Generic driver: • Added missing USART_STOPBITS_0_5 definition for 0.5 stop bit frames. **LL COMP** driver: • LL_COMP_Set{/Get}InputNonInverting() renamed to LL_COMP_Set{/Get}InputMinus. • LL_COMP_Set{/Get}InputInverting() renamed to LL_COMP_Set{/Get}InputPlus. • LL_COMP_Set{/Get}WindowMode() renamed to LL_COMP_Set{/Get}CommonWindowMode().

Date	Revision	Changes
19-Feb-2016	5 (continued)	**LL GPIO** driver: • To align with HAL GPIO, the following GPIO speed definitions have been added: LL_GPIO_SPEED_FREQ_LOW, LL_GPIO_SPEED_FREQ_MEDIUM, LL_GPIO_SPEED_FREQ_HIGH and LL_GPIO_SPEED_FREQ_VERY_HIGH. **LL I2C** driver: • Added new LL_I2C_ConfigFilters() function to configure noise filters. **LL LPTIM** driver: • Added the following new functions: – LL_LPTIM_IsEnabled() – LL_LPTIM_SetWaveform() – LL_LPTIM_SetPolarity() **LL OPAMP** driver: • LL_OPAMP_Get{/Set}PowerRange() renamed into LL_OPAMP_Get{/Set}CommonPowerRange(). **LL SPI** driver: • Removed LL_SPI_Set{/Get}HalfDuplexDirection() functions: this is managed through the TransferDirection parameter in LL_SPI_Set{/Get}TransferDirection(). **LL SWPMI** driver: • Added new LL_SWPMI_IsActivated() function. **LL TIM** driver: • Added the following new functions: – LL_TIM_CC_IsEnabledChannel() – LL_TIM_OC_IsEnabledFast(), LL_TIM_OC_IsEnabledPreload() and LL_TIM_OC_IsEnabledClear() – LL_TIM_IsEnabledMasterSlaveMode() – LL_TIM_EnableExternalClock(), LL_TIM_DisableExternalClock() and LL_TIM_IsEnabledExternalClock() **LL USART** driver: • Added LL_USART_STOPBITS_0_5 definition for usage in LL_USART_Set{/Get}StopBitsLength() and LL_USART_ConfigCharacter();
07-Mar-2017	6	• Added HAL drivers for DCMI interface, supported by USB+LCD line STM32L496xx and USB+LCD+AES line STM32L4A6xx. • Added HAL drivers for HASH, supported by USB+LCD+AES line STM32L4A6xx. • Added HAL and LL drivers for DMA2D controller, supported by USB+LCD line STM32L496xx and USB+LCD+AES line STM32L4A6xx.

Date	Revision	Changes
27-Sep-2017	7	• Added HAL driver for the OctoSPI interface supported by STM32L4+ Series. • Added HAL driver for GFXMMU and LTDC interfaces supported by STM32L4+ Series: STM32L4R7xx (USB_OTG and LCD-TFT Interface line), STM32L4S7xx (USB_OTG, LCD-TFT Interface and AES line), STM32L4R9xx (USB_OTG and MIPI DSIHOST line) and STM32L4S9xx (USB_OTG MPI DSIHOST and AES line). • Added HAL driver for the DSI interface supported by STM32L4+ Series: STM32L4R9xx (USB_OTG and MIPI DSIHOST line) and STM32L4S9xx (USB_OTG, MPI-DSI and AES line).