AN4759
Application note

Using the hardware real-time clock (RTC) in low-power modes with STM32 microcontrollers

Introduction

A real-time clock (RTC) is a computer clock that keeps track of the current time. Although the RTCs are often used in personal computers, servers and embedded systems, they are also present in almost any electronic device that requires an accurate time keeping. The microcontrollers supporting the RTC can be used for chronometers, alarm clocks, watches, small electronic agendas, and many other devices.

This application note describes the features of the real-time clock (RTC) controller embedded in the ultra-low-power STM32 microcontrollers and the steps required to configure the RTC for the use with the calendar, alarm, periodic wakeup unit, tamper detection, timestamp and calibration applications. In this document, the STM32 microcontroller terminology applies to the products listed in *Table 1*.

Software examples are then provided to show how to use the RTC in the low-power modes and how to ensure the tampering detection and timestamp while the main supply is switched off and the MCU is supplied by an alternate battery. One example is showing also a possible implementation of the smooth calibration features.

The X-CUBE-RTC embedded software package is delivered with this application note, containing the source code of the RTC examples and all the embedded software modules required to run the examples.

Table 1. Applicable products

Type	Product series and part number
Microcontrollers	STM32L0 Series, STM32L1 Series, STM32L4 Series, STM32F0 Series, STM32F2 Series, STM32F3 Series, STM32F4 Series, STM32F7 Series, X-CUBE-RTC

Contents

List of tables

List of figures

1 Overview of the STM32 MCU advanced RTC

The real-time clock (RTC) embedded in STM32 MCUs acts as an independent Binary Coded Decimal (BCD) timer/ counter. The RTC can be used to provide a full-featured calendar, alarm, periodic wakeup unit, digital calibration, synchronization, time-stamp, and an advanced tamper detection.

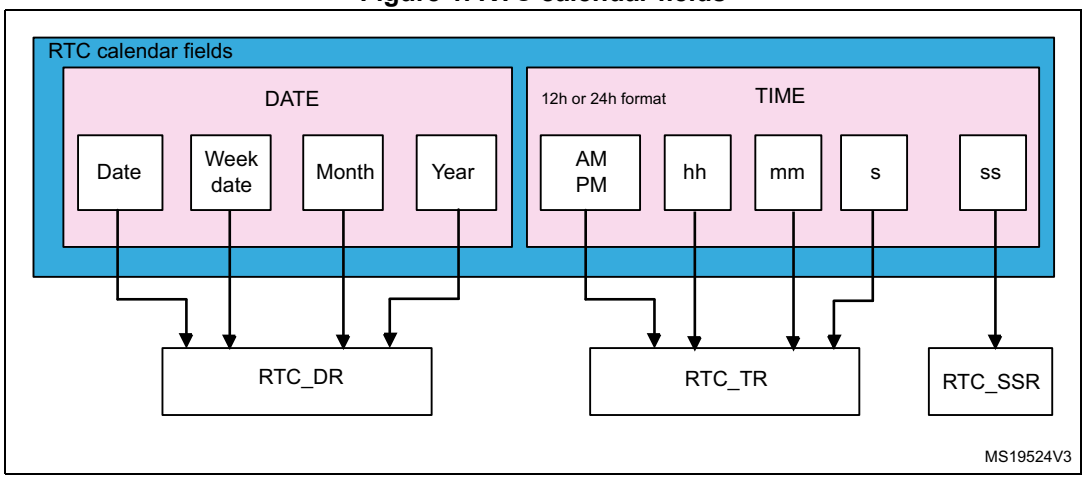
Refer to *Table 14: Advanced RTC features* for the complete list of features available on each device.

1.1 RTC calendar

A calendar keeps track of the time (hours, minutes and seconds) and date (day, week, month, year). The RTC calendar offers several features to easily configure and display the calendar data fields:

- Calendar with:
  - Sub-seconds (not programmable)
  - Seconds
  - Minutes
  - Hours in 12-hour or 24-hour format
  - Day of the week (day)
  - Day of the month (date)
  - Month
  - Year
- Calendar in binary-coded decimal (BCD) format
- Automatic management of 28-, 29- (leap year), 30-, and 31-day months
- Daylight saving time adjustment programmable by software

Figure 1. RTC calendar fields



1. RTC_DR, RTC_TR are RTC Date and Time registers.
2. The sub-second field (RTC_SSR) is the value of the synchronous prescaler's counter. This field is not writable.

1.1.1 Software calendar

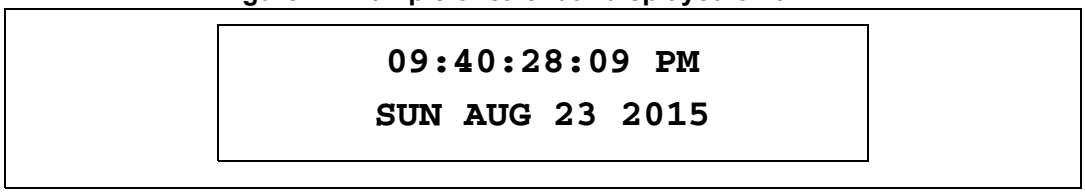
A software calendar can be a software counter (usually 32 bits long) that represents the number of seconds. The software routines convert the counter value to hours, minutes, day of the month, day of the week, month and year. This data can be converted to BCD format and displayed on a standard LCD. Conversion routines use a significant program memory space and are CPU-time consuming, which may be critical in certain real-time applications.

1.1.2 RTC hardware calendar

When using the RTC calendar, the software conversion routines are no longer needed because their functions are performed by hardware.

The STM32 RTC calendar is provided in BCD format. This avoids binary to BCD software conversion routines, which save system resources.

Figure 2. Example of calendar displayed on an LCD



1.1.3 Initializing the calendar

Table 2 describes the steps required to correctly configure the calendar time and date.

Table 2. Steps to initialize the calendar

Step	What to do	How to do it	Comments
1	Disable the RTC registers write protection	Write "0xCA" and then "0x53" into the RTC_WPR register	RTC registers can be modified
2	Enter Initialization mode	Set INIT bit to ‘1’ in RTC_ISR register	The calendar counter is stopped to allow its update
3	Wait for the confirmation of Initialization mode (clock synchronization)	Poll INITF bit of in RTC_ISR until it is set	It takes around 2 RTCCLK clock cycles due to clock synchronization.
4	Program the prescaler values if needed	RTC_PRER register: Write first the synchronous value and then write the asynchronous value	By default, the RTC_PRER prescalers register is initialized to provide 1Hz to the Calendar unit when RTCCLK = 32768Hz
5	Load time and date values in the shadow registers	Set RTC_TR and RTC_DR registers	-
6	Configure the time format (12h or 24h)	Set FMT bit in RTC_CR register	FMT = 0: 24 hour/day format FMT = 1: AM/PM hour format

Table 2. Steps to initialize the calendar (continued)

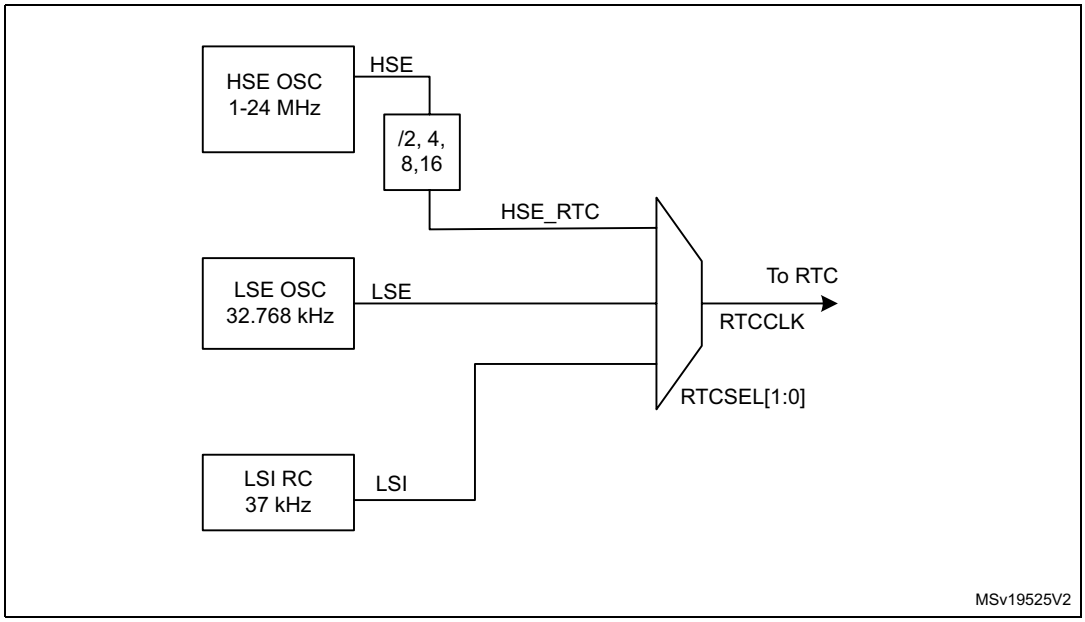
Step	What to do	How to do it	Comments
7	Exit Initialization mode	Clear the INIT bit in RTC_ISR register	The current calendar counter is automatically loaded and the counting restarts after 4 RTCCLK clock cycles
8	Enable the RTC Registers Write Protection	Write "0xFF" into the RTC_WPR register	RTC Registers can no longer be modified

1.1.4 RTC clock configuration

RTC clock source

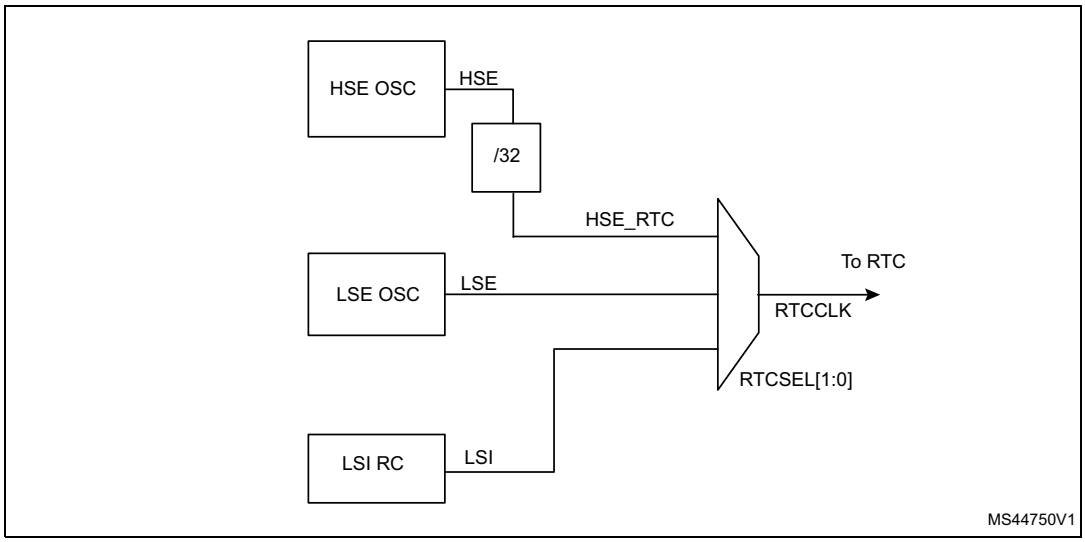
The RTC calendar can be driven by one of three possible clock sources LSE, LSI or HSE (see *Figure 3* and *Figure 4*).

Figure 3. STM32L0 Series, STM32L1 Series, STM32F2 Series and STM32F4 Series RTC clock sources



Note: The RTCSEL[1:0] bits are the RCC Control/status register (RCC_CSR) [17:16] bits.

Figure 4. STM32L4 Series, STM32F0 Series, STM32F3 Series and STM32F7 Series RTC clock sources

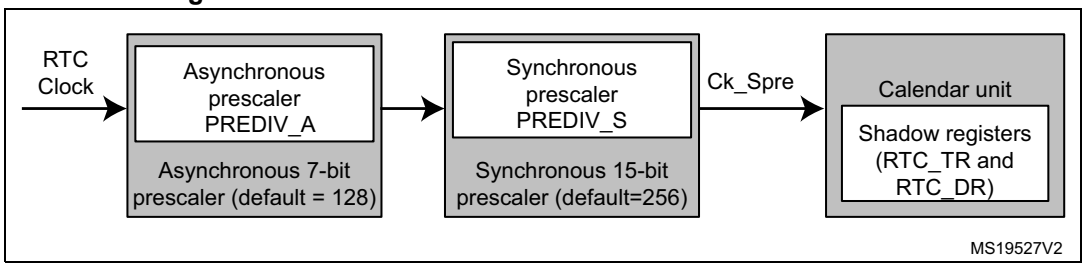


Note: The RTCSEL[1:0] bits are the backup domain control register (RCC_BDCR) [9:8] bits.

How to adjust the RTC calendar clock

The RTC features several prescalers that allow delivering a 1 Hz clock to the calendar unit, regardless of the clock source.

Figure 5. Prescalers from RTC clock source to calendar unit



The formula to calculate ck_spre is:

$$ck\_spre = \frac{RTCCLK}{(PREDIV\_A + 1) \times (PREDIV\_S + 1)}$$

Where:

- RTCCLK can be any clock source: HSE_RTC, LSE or LSI
- PREDIV_A can be 1,2,3,..., or 127
- PREDIV_S can be 0,1,2,..., or 32767

Table 3 shows several ways to obtain the calendar clock (ck_spre) = 1 Hz.

Table 3. Calendar clock equal to 1 Hz with different clock sources(1)

RTCCLK Clock source	Prescalers		ck_spre
	PREDIV_A[6:0]	PREDIV_S[14:0]	
HSE_RTC = 1 MHz	124 (div 125)	7999 (div 8000)	1 Hz
LSE = 32.768 kHz	127 (div 128)	255 (div 256)	1 Hz
LSI = 32 kHz	127 (div 128)	249 (div 250)	1 Hz
LSI = 37 kHz	124 (div 125)	295 (div 296)	1 Hz
LSI = 40 kHz	127 (div 128)	311 (div 312)	1 Hz

1. Other PREDIV_A[6:0]/PREDIV_S[14:0] values are possible. The user must always prefer the combination where PREDIV_A[6:0] is the highest for the needed accuracy.

1.2 RTC alarms

1.2.1 RTC alarm configuration

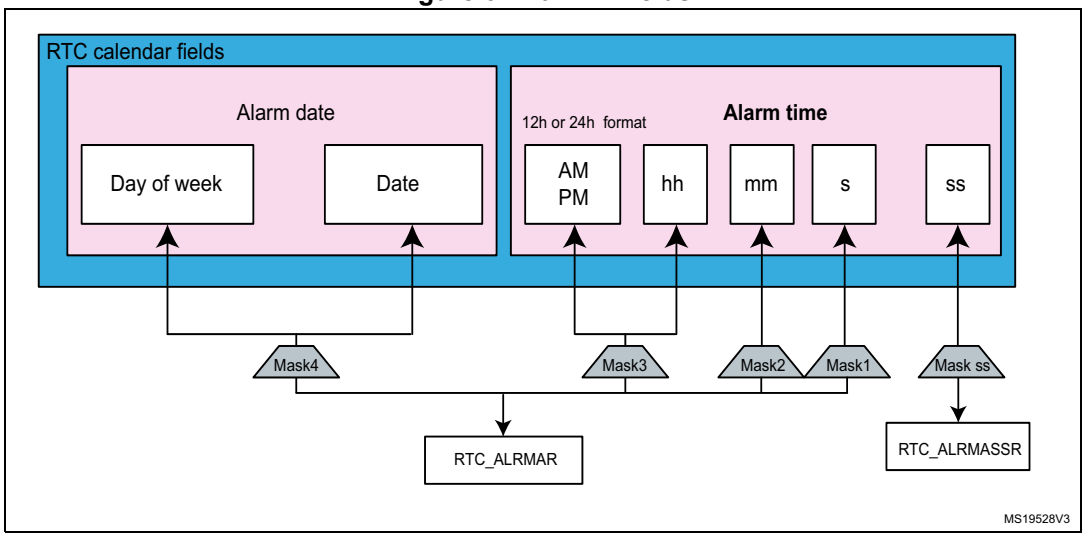
The RTC embeds two alarms, alarm A and alarm B, which are similar. An alarm can be generated at a given time or/and date programmed by the user.

The RTC provides a rich combination of alarms settings, and offers many features to make it easy to configure and display these alarms settings.

Each alarm unit provides the following features:

- Fully programmable alarm: sub-second (this is discussed later), seconds, minutes, hours and date fields can be independently selected or masked to provide a rich combination of alarms.
- Ability to wake up the device from low power modes when the alarm occurs.
- The alarm event can be routed to a specific output pin with configurable polarity.
- Dedicated alarm flags and interrupt.

Figure 6. Alarm A fields



1. RTC_ALRMAR is an RTC register. The same fields are also available for the RTC_ALRMBR register.
2. RTC_ALRMASSR is an RTC register. The same field is also available for the RTC_ALRMBSSR register.
3. Maskx are the bits in the RTC_ALRMAR register that enable/disable the RTC_ALARM fields used for alarm A. For more details, refer to *Table 5*.
4. Mask ss are the bits in the RTC_ALRMASSR register.

The time configuration bit-fields of the alarm configuration register are mapped into the same bit offsets as those on the RTC_TR time register for easier software manipulation. When the RTC time counter reaches the value programmed in the alarm register, a flag is set to indicate that an alarm event occurred.

The RTC alarm can be configured by hardware to generate different types of alarms. For more details, refer to *Table 5*.

Programming the alarm

Table 4 describes the steps required to configure alarm A.

Table 4. Steps to configure the alarm

Step	What to do	How to do it	Comments
1	Disable the RTC registers Write protection	Write "0xCA" and then "0x53" into the RTC_WPR register.	RTC registers can be modified
2	Disable alarm A	Clear ALRAE(1) bit in RTC_CR register(2).	-
3	Check that the RTC_ALRMAR register can be accessed	Poll ALRAWF(3) bit until it is set in RTC_ISR.	It takes around 2 RTCCLK clock cycles due to clock synchronization.
4	Configure the alarm	Configure RTC_ALRMAR(4) register.	The alarm hour format must be the same(5) as the RTC Calendar in RTC_ALRMAR.
5	Re-enable alarm A	Set ALRAE(6) bit in RTC_CR register.	-
6	Enable the RTC registers Write protection	Write "0xFF" into the RTC_WPR register.	RTC registers can no longer be modified

1. Respectively ALRBE bit for alarm B.
2. RTC alarm registers can only be written when the corresponding RTC alarm is disabled or during RTC Initialization mode.
3. Respectively ALRBWF bit for alarm B.
4. Respectively RTC_ALRMBR register for alarm B.
5. As an example, if the alarm is configured to occur at 3:00:00 PM, the alarm will not occur even if the calendar time is 15:00:00, because the RTC calendar is 24-hour format and the alarm is 12-hour format.
6. Respectively ALRBE bit for alarm B.

Configuring the alarm behavior using the MSKx bits

The alarm behavior can be configured using the MSKx bits (x = 1, 2, 3, 4) of the RTC_ALRMAR register for alarm A (RTC_ALRMBR register for alarm B).

Table 5 shows all the possible alarm settings. As an example, to configure the alarm time to 23:15:07 on monday (assuming that the WDSEL = 1), configure the RTC_ALRMAR register (resp. RTC_ALRMBR register) with the MSKx bits set to 0000b. When the WDSEL = 0, all the cases are similar, except that the Alarm Mask field compares with the day number and not the day of the week, and MSKx bits must be set to 0000b.

Table 5. Alarm combinations

MSK3	MSK2	MSK1	MSK0	Alarm behavior
0	0	0	0	**All the fields are used in the alarm comparison:** The alarm occurs at 23:15:07, each Monday.
0	0	0	1	**The seconds do not matter in the alarm comparison** The alarm occurs every second of 23:15, each Monday.
0	0	1	0	**The minutes do not matter in the alarm comparison** The alarm occurs at the 7th second of every minute of 23:XX, each Monday.

Table 5. Alarm combinations (continued)

MSK3	MSK2	MSK1	MSK0	Alarm behavior
0	0	1	1	**The minutes and seconds do not matter in the alarm comparison**
0	1	0	0	**The hours do not matter in the alarm comparison**
0	1	0	1	**The hours and seconds do not matter in the alarm comparison**
0	1	1	0	**The hours and minutes do not matter in the alarm comparison**
0	1	1	1	**The hours, minutes and seconds do not matter in the alarm comparison** The alarm is set every second, each Monday, during the whole day.
1	0	0	0	**The week day (or date, if selected) do not matter in the alarm comparison** The alarm occurs all the days at 23:15:07.
1	0	0	1	**The week day and seconds do not matter in the alarm comparison**
1	0	1	0	**The week day and minutes do not matter in the alarm comparison**
1	0	1	1	**The week day, minutes and seconds do not matter in the alarm comparison**
1	1	0	0	**The week day and hours do not matter in the alarm comparison**
1	1	0	1	**The week day, hours and seconds do not matter in the alarm comparison**
1	1	1	0	**The week day, hours and minutes do not matter in the alarm comparison**
1	1	1	1	**The alarm occurs every second**

Caution: If the second field is selected (MSK0 bit reset in RTC_ALRMAR or RTC_ALRMBR), the synchronous prescaler division factor PREDIV_S set in the RTC_PRER register must be at least 3 to ensure a correct behavior.

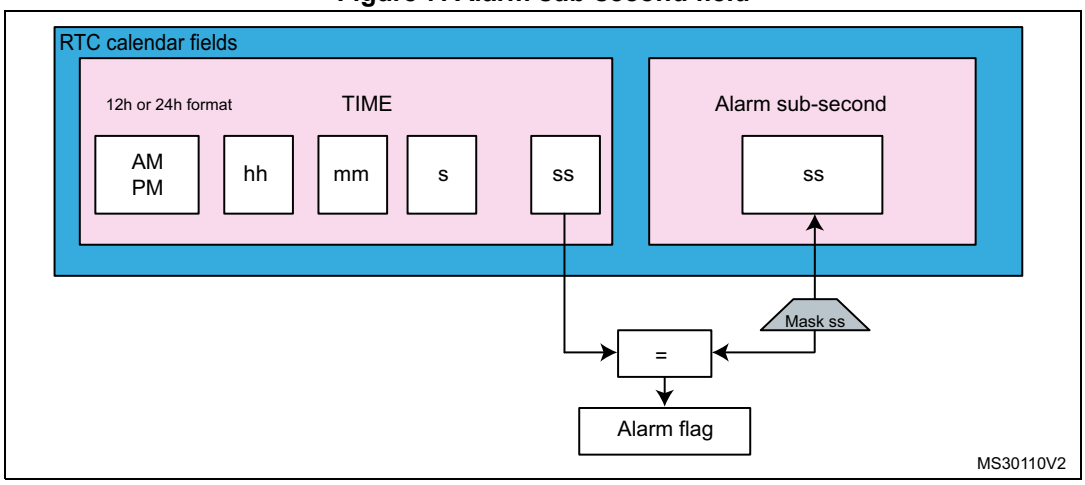
1.2.2 Alarm sub-second configuration

The RTC unit provides programmable alarms, sub-second A and B, which are similar. They generate alarms with a high resolution (for the second division).

The value programmed in the alarm sub-second register is compared to the content of the sub-second field in the calendar unit.

The sub-second field counter counts down from the value configured in the synchronous prescaler to zero, and then reloads a value from the RTC_SPRE register.

Figure 7. Alarm sub-second field



Note: Mask ss is the most significant bit in the sub-second alarm. This bit is compared to the synchronous prescaler register.

The alarm sub-second can be configured using the mask ss bit in the alarm sub-second register. *Table 6: Alarm sub-second mask combinations* shows the configuration possibilities for the mask register and provides an example with the following settings:

- Select LSE as the RTC clock source (for example LSE = 32768 Hz).
- Make sure that the asynchronous prescaler is set at 127.
- Make sure that the synchronous prescaler is set at 255 (the calendar clock is equal to 1Hz).
- Set the alarm A sub-second to 255 (put 255 in the SS[14:0] field).

Table 6. Alarm sub-second mask combinations

MASK SS	Alarm A sub-second behavior	Example result
0	There is no comparison on sub-second for alarm. The alarm is activated when the second unit is incremented.	The alarm is activated every 1 second
1	Only the AlarmA_SS[0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every (1/128) s
2	Only the AlarmA_SS[1:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every (1/64) s
3	Only the AlarmA_SS[2:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every (1/32) s
4	Only the AlarmA_SS[3:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every (1/16) s
5	Only the AlarmA_SS[4:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 125 ms
6	Only the AlarmA_SS[5:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 250 ms
7	Only the AlarmA_SS[6:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 500 ms
8	Only the AlarmA_SS[7:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 1 s
9	Only the AlarmA_SS[8:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 1 s
10	Only the AlarmA_SS[9:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 1 s
11	Only the AlarmA_SS[10:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 1 s
12	Only the AlarmA_SS[11:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 1 s
13	Only the AlarmA_SS[12:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 1 s
14	Only the AlarmA_SS[13:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 1 s
15	Only the AlarmA_SS[14:0] bit is compared to the RTC sub-second register RTC_SSR	The alarm is activated every 1 s

Note: The overflow bit in the sub-second register (bit15) is never compared.

1.3 RTC periodic wakeup unit

The STM32 MCUs provide several low-power modes to reduce the power consumption. The RTC features a periodic timebase and wakeup unit able to wake up the system from low-power modes. This unit is a programmable down-counting auto-reload timer. When this counter reaches zero, a flag and an interrupt (if enabled) are generated.

The wakeup unit has the following features:

- Programmable down-counting auto-reload timer.
- Specific flag and interrupt capable of waking up the device from low-power modes.
- Wakeup alternate function output which can be routed to RTC_ALARM output (unique pad for alarm A, alarm B or Wakeup events) with configurable polarity.
- A full set of prescalers to select the desired waiting period.

1.3.1 Programming the auto-wakeup unit

Table 7 describes the steps required to configure the auto-wakeup unit.

Table 7. Steps to configure the auto-wakeup unit

Step	What to do	How to do it	Comments
1	Disable the RTC registers Write protection	Write "0xCA" and then "0x53" into the RTC_WPR register	RTC registers can be modified
2	Disable the wakeup timer.	Clear WUTE bit in RTC_CR register	
3	Ensure access to Wakeup auto-reload counter and bits WUCKSEL[2:0] is allowed.	Poll WUTWF until it is set in RTC_ISR	It takes around 2 RTCCLK clock cycles due to clock synchronization
4	Program the value into the wakeup timer.	Set WUT[15:0] in RTC_WUTR register	See *Section 1.3.2: Maximum and minimum RTC wakeup period*
5	Select the desired clock source.	Program WUCKSEL[2:0] bits in RTC_CR register	
6	Re-enable the wakeup timer.	Set WUTE bit in RTC_CR register	The wakeup timer restarts counting down
7	Enable the RTC registers Write protection	Write "0xFF" into the RTC_WPR register	RTC registers can no more be modified

1.3.2 Maximum and minimum RTC wakeup period

The wakeup unit clock is configured through the WUCKSEL[2:0] bits of RTC_CR1 register. Three different configurations are possible:

- Configuration 1: WUCKSEL[2:0] = 0xxb for short wakeup periods (see *Periodic timebase/wakeup configuration for clock configuration 1*)
- Configuration 2: WUCKSEL[2:0] = 10xb for medium wakeup periods (see *Periodic timebase/wakeup configuration for clock configuration 2*)
- Configuration 3: WUCKSEL[2:0] = 11xb for long wakeup periods (see *Periodic timebase/wakeup configuration for clock configuration 3*)

Periodic timebase/wakeup configuration for clock configuration 1

Figure 8 shows the prescaler connection to the timebase/wakeup unit and *Table 8* gives the timebase/wakeup clock resolutions corresponding to configuration 1.

The prescaler depends on the wakeup clock selection:

- WUCKSEL[2:0] =000: RTCCLK/**16** clock is selected
- WUCKSEL[2:0] =001: RTCCLK/**8** clock is selected
- WUCKSEL[2:0] =010: RTCCLK/**4** clock is selected
- WUCKSEL[2:0] =011: RTCCLK/**2** clock is selected

Figure 8. Prescalers connected to the timebase/wakeup unit for configuration 1

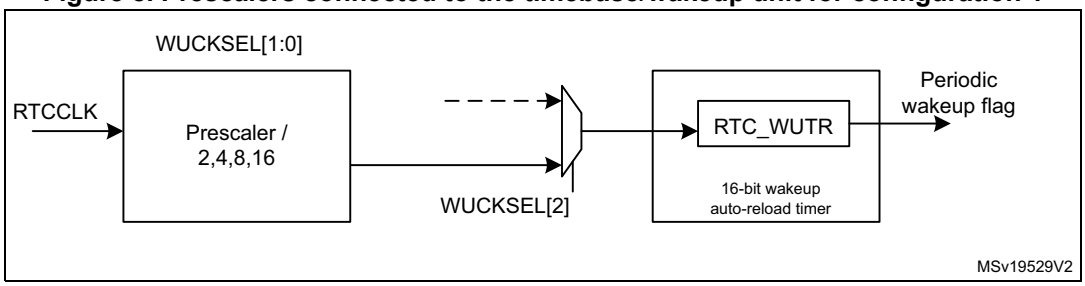


Table 8. Timebase/wakeup unit period resolution with clock configuration 1

Clock source	Wakeup period resolution	
	WUCKSEL[2:0] = 000b (div16)	**WUCKSEL[2:0] = 011b (div2)**
LSE = 32 768 Hz	488.28 µs	61.035 µs

When RTCCLK= 32768 Hz, the minimum timebase/wakeup resolution is 61.035 µs, and the maximum resolution is 488.28 µs. As a result:

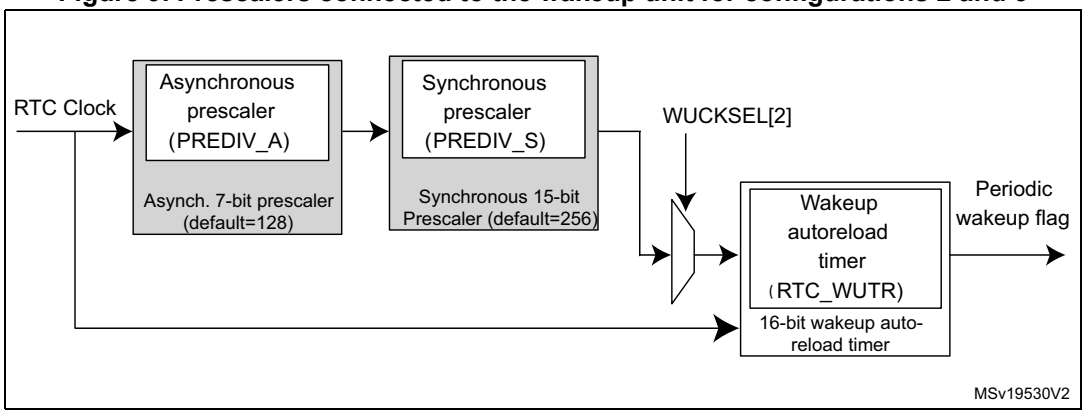
- The minimum timebase/wakeup period is (0x0001 + 1) x 61.035 µs = 122.07 µs.

  The timebase/wakeup timer counter WUT[15:0] cannot be set to 0x0000 with WUCKSEL[2:0]=011b ($f_{RTCCLK}/2$) because this configuration is prohibited. *Refer to the STM32Lx reference manuals for more details*.
- The maximum timebase/wakeup period is (0xFFFF+ 1) x 488.28 µs = 32 s.

Periodic timebase/wakeup configuration for clock configuration 2

Figure 9 shows the prescaler connection to the timebase/wakeup unit corresponding to configuration 2 and 3.

Figure 9. Prescalers connected to the wakeup unit for configurations 2 and 3



If ck_spre (synchronous prescaler output clock) is adjusted to 1 Hz, then:

- The minimum timebase/wakeup period is (0x0000 + 1) x 1 s = 1 s.
- The maximum timebase/wakeup period is (0xFFFF+ 1) x 1 s = 65536 s (18 hours).

Periodic timebase/wakeup configuration for clock configuration 3

For this configuration, the resolution is the same as for configuration 2. However, the timebase/wakeup counter downcounts starting from **0x1FFFF** to 0x00000, instead of **0xFFFF** to 0x0000 for configuration 2.

If ck_spre is adjusted to 1 Hz, then:

- The minimum timebase/wakeup period is:
  (0x10000 + 1) x 1 s = 65537 s (18 hours + 1 s)
- The maximum timebase/wakeup period is:
  (0x1FFFF+ 1) x 1 s = 131072 s (36 hours).

Summary of timebase/wakeup period extrema

When RTCCLK= 32768 Hz and ck_spre (Synchronous prescaler output clock) is adjusted to 1 Hz, the minimum and maximum period values are listed in *Table 9*.

Table 9. Min. and max. timebase/wakeup period when RTCCLK= 32768

Configuration	Minimum period	Maximum period
1	122.07 µs	32 s
2	1 s	18 hours
3	18 hours +1 s	36 hours

1.4 Digital smooth calibration

1.4.1 RTC calibration basics

The term “quartz-accurate” has become a familiar phrase used to describe the accuracy of many time keeping functions. The quartz oscillators provide an accuracy far superior to that of other conventional oscillator designs, but they are not perfect. The quartz crystals are sensitive to temperature variations. *Figure 10* shows the relationship between accuracy (acc), temperature (T) and curvature (K) for a typical 32.768 kHz crystal. The curve follows the general formula given below:

$acc = K \times (T-To)^2$, where:

- To = 25 °C ± 5 °C
- K = –0.032 ppm/°C^2

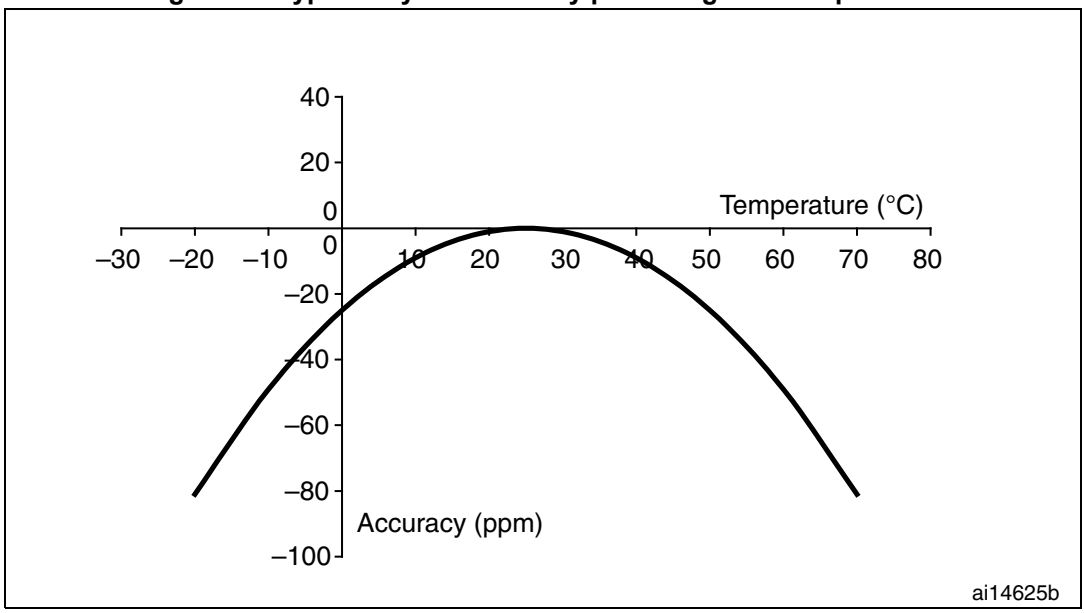
Note: The variable K is crystal-dependent, the value indicated here is for the crystal mounted on the STM32L476RG-Nucleo board. Refer to the crystal manufacturer for more details on this parameter.

The clocks used in most applications require a high degree of accuracy, and there are several factors involved in achieving this accuracy.

Two major approaches exist to compensate for an oscillator's oscillation frequency deviation when the generated signal is used to clock a time-keeping logic:

- The analog approach where the oscillator is built with embedded capacitor banks on both of the oscillator's input and output. Thus, to adjust the oscillation frequency, the software needs to configure the oscillator to switch on or off some of the embedded load capacitors to adjust the overall load capacitance seen by the crystal resonator. It must be noted that the two external load capacitors are always needed even with this kind of oscillators. The two external load capacitors make the dominant part of the crystal resonator's load capacitance. The oscillator's embedded capacitor banks are only intended to compensate for the capacitance value dispersion of the two external load capacitors or to compensate for the temperature variation. This approach suffers from most of drawbacks that generally analog circuits suffer from, like relatively important value dispersion from one chip to another, etc.
- The digital approach, where the deviation of the oscillation frequency is not compensated for at the oscillator level. Instead of that, the compensation is made at the time-keeping logic/function level. The time-keeping logic either adds or removes few clock cycles to/from the deviating clock signal that is feeding its counter. The chief advantage of this approach is the deterministic compensation effect from one device to another. This approach is adopted to compensate for the LSE oscillation frequency deviation when it is used to clock the RTC peripheral. The RTC peripheral is built with a dedicated register to configure the number of clock cycles to add or remove from the feeding clock signal.

Figure 10. Typical crystal accuracy plotted against temperature

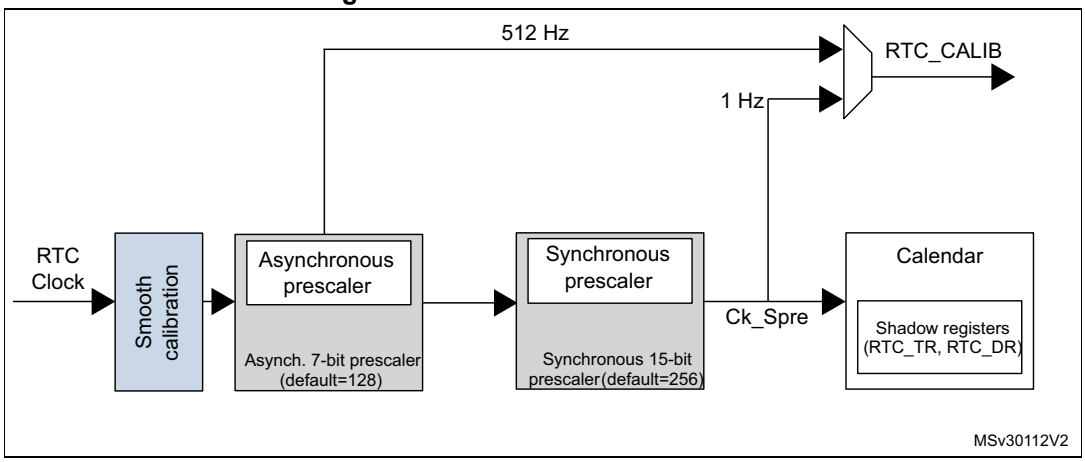


1.4.2 Methodology

The RTC clock frequency can be corrected using a series of small adjustments by adding or subtracting individual RTCCLK pulses.The RTC clock can be calibrated with a resolution of about 0.954 ppm with a range from -487.1 ppm to +488.5 ppm.

This digital smooth calibration is designed to compensate for the inaccuracy of crystal oscillators due to the temperature, crystal aging.

Figure 11. Smooth calibration block



The user can compute the clock deviation using the RTC_CALIB signal, then update the calibration block. It is also possible to input an external 1 Hz reference and make the adequate processing inside the MCU to correct the RTC clock. The user finds such example in *Section 5.3: Application example project*.

It is possible to check the calibration result using the calibration output 512 Hz or 1 Hz for the RTC_CALIB signal. Refer to *Table 14: Advanced RTC features*.

A smooth calibration consists of masking and adding N (configurable) 32 kHz-frequency pulses that are well distributed in a configurable window (8 s, 16 s or 32 s).

The number of masked or added pulses is defined using CALP and CALM in the RTC_CALR register.

By default, the calibration window is 32 seconds. It can be reduced to 8 or 16 seconds by setting the CALW8 bit or the CALW16 bit in the RTC_CALR register:

Note: The 0.954 PPM accuracy of the smooth calibration is only achievable by 32 s calibration window, for 16 s the best accuracy is (0.954 x 2) and for 8 s is (0.954 x 4).

Example 1: setting CALM[0] to 1, CALP=0 and using 32 seconds as a calibration window results in exactly one pulse being masked for 32 seconds.

Example 2: setting CALM[2] to 1, CALP=0 and using 32 seconds as a calibration window results in exactly 4 pulses being masked for 32 seconds.

Note: Both the CALM and CALP can be used and, in this case, an offset ranging from -511 to +512 pulses can be added for 32 seconds (calibration window).

When the asynchronous prescaler is less than 3, CALP cannot be set to 1.

The formula to calculate the effective calibrated frequency (FCAL), given the input frequency (FRTCCLK), is:

$F_{CAL} = F_{RTCCLK} \times [1 + (CALP \times 512 - CALM) / (2^{20} + CALM - CALP \times 512)]$.

A smooth calibration can be performed on the fly so that it can be changed when the temperature changes or if other factors are detected.

Checking the smooth calibration

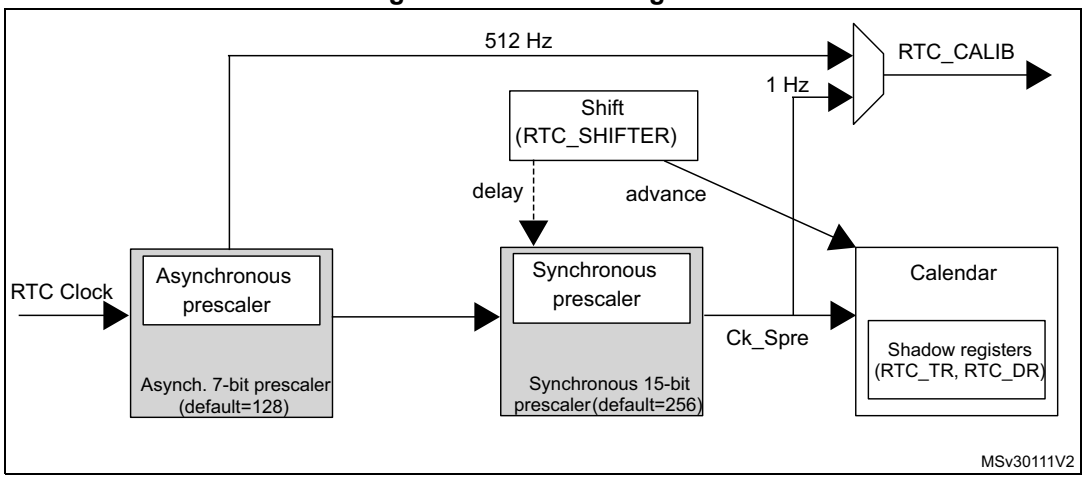
The smooth calibration effect on the calendar clock (RTC Clock) can be checked by:

- Calibration using the RTC_CALIB output (1 Hz).
- Calibration using the sub-second alarms.
- Calibration using the wakeup timer.

1.5 Synchronizing the RTC

The RTC calendar can be synchronized to a more precise clock, “remote clock”, using the RTC shift feature. After reading the RTC sub-second field, a calculation of the precise offset between the time being maintained by the remote clock and the RTC can be made. The RTC can be adjusted by removing this offset with a fine adjustment using the shift register control.

Figure 12. RTC shift register



It is not possible to check the “Synchronization” shift function using the RTC_CALIB output since the shift operation has no impact on the RTC clock, other than adding or subtracting a few fractions from the calendar counter.

Correcting the RTC calendar time

If the RTC clock is advanced compared to the remote clock by n fractions of seconds, the offset value must be written in SUBFS, which will be added to the synchronous prescaler’s counter. As this counter counts down, this operation effectively subtracts from (delays) the clock by:

Delay (seconds) = SUBFS / (PREDIV_S + 1)

If the RTC is delayed compared to the remote clock by n fractions of seconds, the offset value can effectively be added to the clock (advancing the clock) when the ADD1S function is used in conjunction with SUBFS, effectively advancing the clock by:

Advance (seconds) = (1 - (SUBFS / (PREDIV_S + 1))).

1.6 RTC reference clock detection

The reference clock (at 50 Hz or 60 Hz) should have a higher precision than the 32.768 kHz LSE clock. This is why the RTC provides a reference clock input (RTC_REFIN pin) that can be used to compensate the imprecision of the calendar frequency (1 Hz).

The RTC_REFIN pin should be configured in input floating mode.

This mechanism enables the calendar to be as precise as the reference clock.

The reference clock detection is enabled by setting the REFCKON bit of the RTC_CR register.

When the reference clock detection is enabled, PREDIV_A and PREDIV_S must be set to their default values: PREDIV_A = 0x007F and PREDIV_S = 0x00FF.

Note: This feature is only valid with the RTC clock from LSE oscillator oscillating at 32.768 KHz due to this requirement.

When the reference clock detection is enabled, each 1 Hz clock edge is compared to the nearest reference clock edge (if one is found within a given time window). In most cases, the two clock edges are properly aligned. When the 1 Hz clock becomes misaligned due to the imprecision of the LSE clock, the RTC shifts the 1 Hz clock a bit so that future 1 Hz clock edges are aligned. The update window is 3 ck_apre periods.

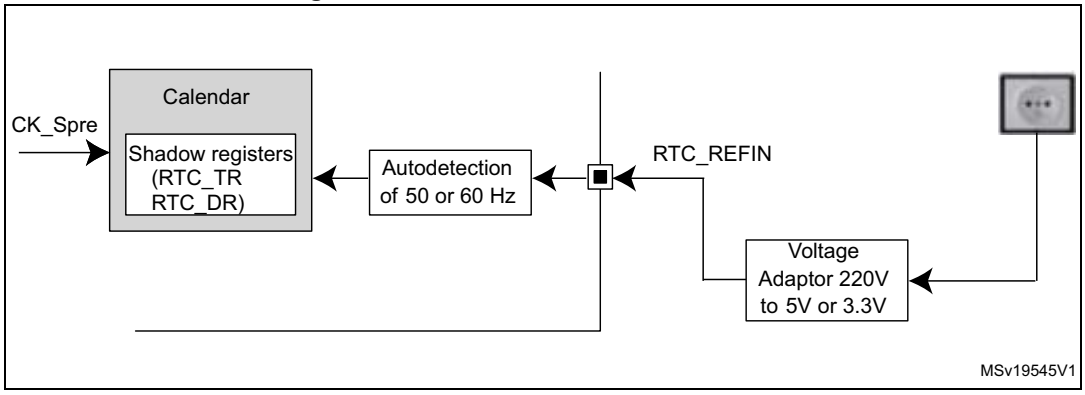
If the reference clock halts, the calendar is updated continuously based solely on the LSE clock. The RTC then waits for the reference clock using a detection window centered on the Synchronous Prescaler output clock (ck_spre) edge. The detection window is 7 ck_apre periods.

The reference clock can have a large local deviation (for instance in the range of 500 ppm), but in the long term it must be much more precise than 32 kHz quartz.

The detection system is used only when the reference clock needs to be detected back after a loss. As the detection window is a bit larger than the reference clock period, this detection system brings an uncertainty of 1 ck_ref period (20 ms for a 50 Hz reference clock) because it is possible to have 2 ck_ref edges in the detection window. Then the update window is used, which brings no error as it is smaller than the reference clock period.

Assuming that ck_ref is not lost more than once a day. So the total uncertainty per month would be 20 ms x 1x 30 = 0.6 s, which is much less than the uncertainty of a typical quartz (53 s/month for +/-20 ppm quartz).

Figure 13. RTC reference clock detection



Note: The reference clock calibration and the RTC synchronization (shift feature) cannot be used together.

The reference clock calibration is the best (ensures a high calibrated time) if the 50 Hz is always available. If the 50 Hz input is lost, the RTC accuracy is provided by the LSE crystal.

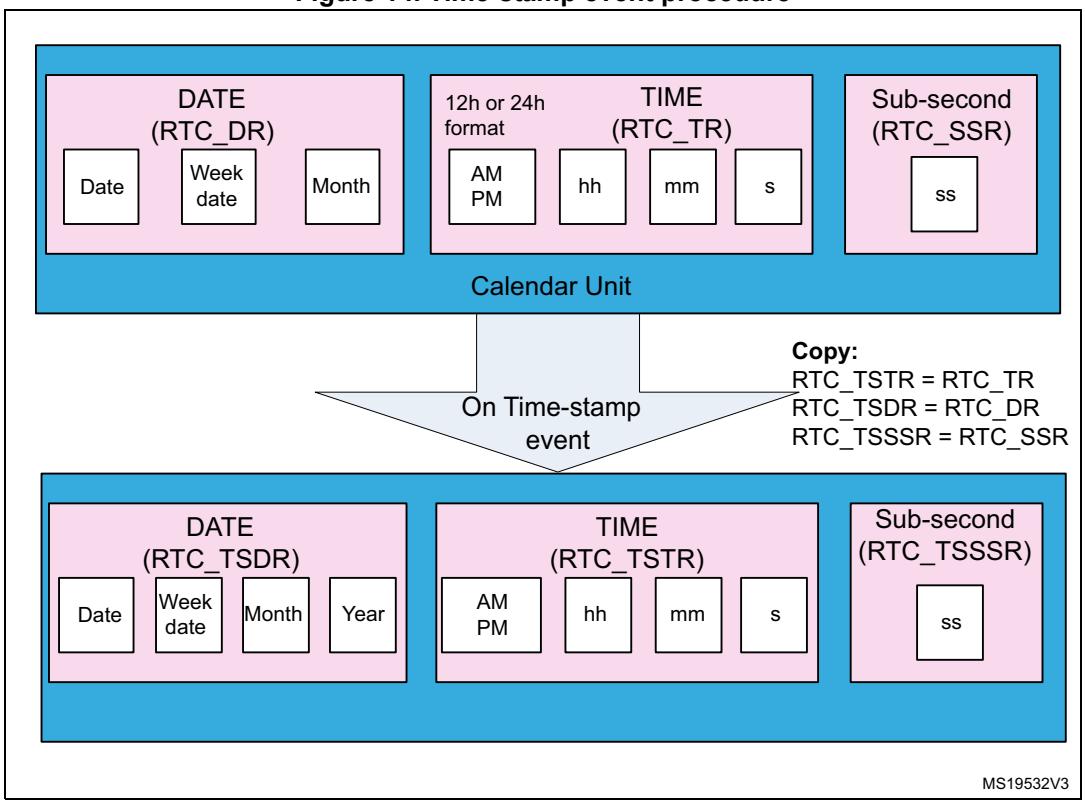
The reference clock detection cannot be used in Vbat mode.

The reference clock calibration can only be used if the user provides a precise 50 or 60 Hz input.

1.7 Time-stamp function

The Time-stamp feature provides the means to automatically save the current calendar when some specific events occur.

Figure 14. Time-stamp event procedure



Provided that the time-stamp function is enabled, the calendar is saved in the time-stamp registers (RTC_TSTR, RTC_TSDR, RTC_TSSSR) when an internal or external time-stamp event is detected. When a time-stamp event occurs, the time-stamp flag bit (TSF) in the RTC_ISR register is set.

The events that can generate a timestamp are:

- An edge detection on the RTC_TS I/O
- A tamper event detection (from all RTC_TAMP I/Os)
- A switch to VBAT when the main supply if powered off (STM32L4x devices only)

Table 10. Time-stamp features

What to do	How to do it	Comments
Enable Time-stamp	Setting the TSE or ITSE bit of RTC_CR register to 1	ITSE is only available on L4
Detect a time-stamp event by interrupt	Setting the TSIE bit in the RTC_CR register	An interrupt is generated when a time-stamp event occurs.
Detect a time-stamp event by polling	By polling on the time-stamp flags (TSF or ITSF(1)) in the RTC_ISR register	To clear the flag, write zero on the TSF bit or ITSF.(2)
Detect a Time-stamp overflow event(3)	By checking on the time-stamp over flow flag (TSOVF(4)) in the RTC_ISR register.	– To clear the flag, write zero in the TSOVF bit. – Time-stamp registers (RTC_TSTR and RTC_TSDR, RTC_TSSSR) maintain the results of the previous event. – If a time-stamp event occurs immediately after the TSF bit is supposed to be cleared, then both TSF and TSOVF bits are set.

1. TSF is set 2 ck_apre cycles after the time-stamp event occurs due to the synchronization process.
2. To avoid masking a time-stamp event occurring at the same moment, the application must not write ‘0’ into TSF bit unless it has already read it to‘1’.
3. The time-stamp overflow event is not connected to an interrupt.
4. There is no delay in the setting of TSOVF. This means that if two time-stamp events are close to each other, TSOVF can be seen as '1' while TSF is still '0'. As a consequence, it is recommended to poll TSOVF only after TSF has been set.

1.8 RTC tamper detection function

The RTC includes up to 3 tamper detection inputs. The tamper input active level/edge can be configured and each one has an individual flag (TAMPxF bit in RTC_ISR register).

A tamper detection event generates an interrupt when the TAMPIE or TAMPxIE bit in RTC_TAMPCR register is set.

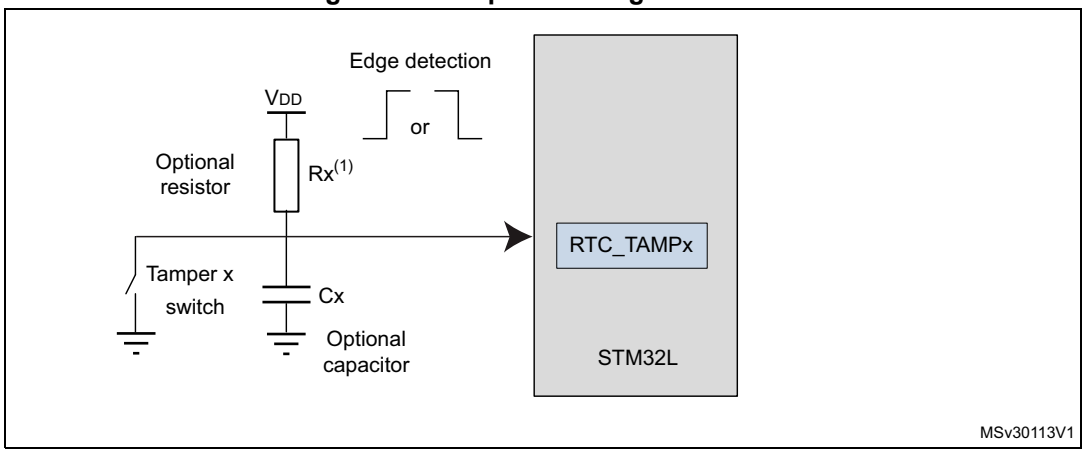
The configuration of the tamper filter, “TAMPFLT bits”, defines whether the tamper detection is activated on edge (set TAMPFLT to “00“), or on level (TAMPFLT must be different from “00“).

Note: The number of tamper inputs depends on product packages. Each input has a “TAMPxF” individual flag in the RTC_TAMP register.

1.8.1 Edge detection on tamper input

When the TAMPFLT bits are set to zero, the tamper input detection triggers when a rising edge or a falling edge (depending on the TAMPxTRG bit) is observed on the corresponding RTC_TAMPx input pin.

Figure 15. Tamper with edge detection



1. If the power consumption is not a concern, the internal 40 kΩ pull-up resistor can be used.

Note: With the edge detection, the sampling and precharge features are deactivated.

Table 11. Tamper features (edge detection)

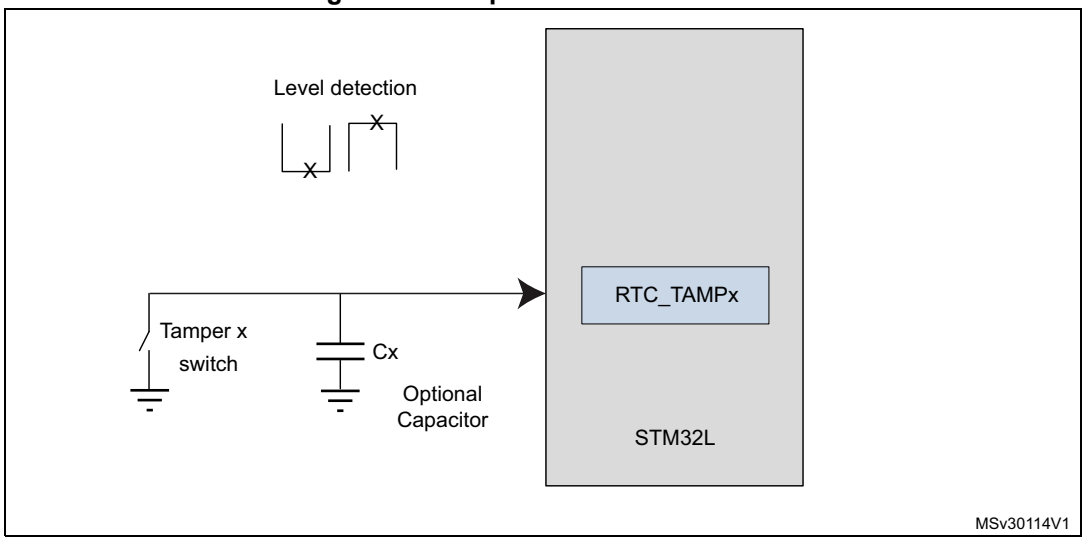
What to do	How to do it	Comments
Enable Tamper	Set the TAMPxE bit of RTC_TAMPCR register to 1	-
Select Tamper active edge detection	Select with TAMPxTRG bit in RTC_TAMPCR register	The default edge is rising edge.
Detect a Tamper event by interrupt	Set the TAMPIE or TAMPxIE bit in the RTC_TAMPCR register	An interrupt is generated when a tamper detection event occurs.
Detect a Tamper event by polling	Poll the time-stamp flag (TAMPxF) in the RTC_ISR register	To clear the flag, write zero in the TAMPxF bit.

1.8.2 Level detection on tamper input

Setting the tamper filter “TAMPFLT” to a value other than zero means that the tamper input triggers when a selected level (high or low) is observed on the corresponding RTC_TAMPx input pin.

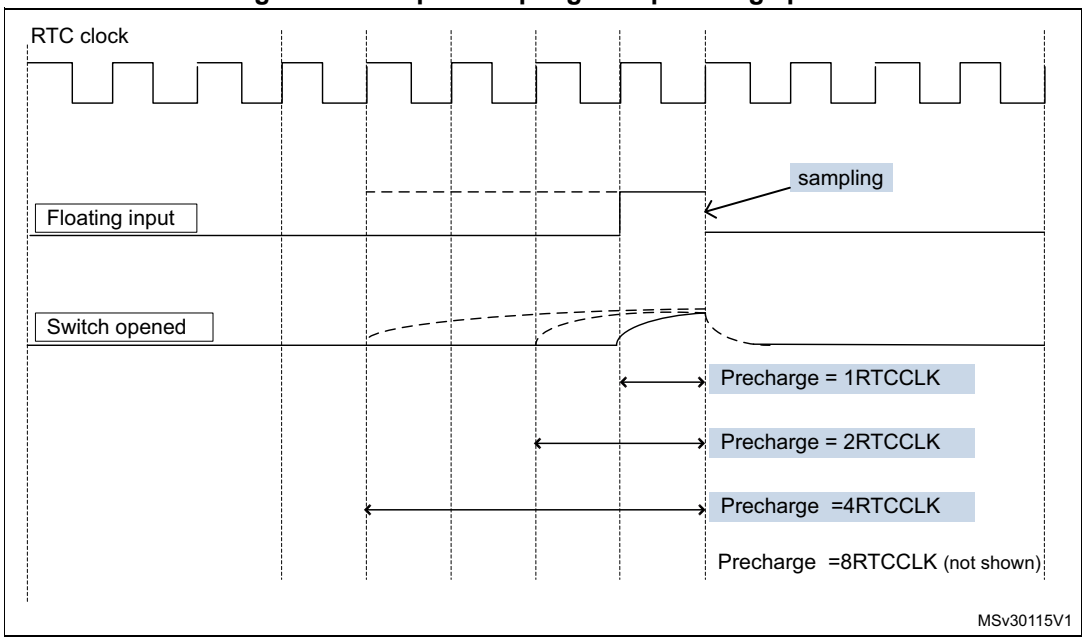
A tamper detection event is generated when either 2, 4 or 8 (depending on the TAMPFLT value) consecutive samples are observed at the selected level.

Figure 16. Tamper with level detection



Using the level detection (tamper filter set to a non-zero value), the tamper input pin can be precharged by resetting TAMPPUDIS through an internal resistance before sampling its state. In order to support the different capacitance values, the length of the pulse during which the internal pull-up is applied can be 1, 2, 4 or 8 RTCCLK cycles.

Figure 17. Tamper sampling with precharge pulse



Note: When the internal pull-up is not applied, the I/Os Schmitt triggers are disabled in order to avoid an extra consumption if the tamper switch is open.

The trade-off between the tamper detection latency and the power consumption through the weak pull-up or external pull-down can be reduced by using a tamper sampling frequency feature. The tamper sampling frequency is determined by configuring the TAMPFREQ bits in the RTC_TAMPCR register. When using the LSE at 32768 Hz as the RTC clock source, the sampling frequency can be 1, 2, 4, 8, 16, 32, 64, or 128 Hz.

Table 12. Tamper features (level detection)

What to do	How to do it	Comments
Enable Tamper	Set the TAMPxE bit of RTC_TAMPCR register to 1	-
Configure Tamper filter count	Configure TAMPFLT bits in RTC_TAMPCR register	-
Configure Tamper sampling frequency	Configure TAMPFREQ bits in RTC_TAMPCR register	Default value is 1Hz
Configure tamper internal pull-up and precharge duration	Configure TAMPPUDIS and TAMPPPRCH bits in RTC_TAMPCR register	-
Select Tamper active edge/Level detection	Select with TAMPxTRG bit in RTC_TAMPCR register	Edge or Level is depending on tamper filter configuration.
Detect a Tamper event by interrupt	Set the TAMPIE or TAMPxIE bit in the RTC_TAMPCR register	An interrupt is generated when tamper detection event occurs.
Detect a Tamper event by polling	Poll the time-stamp flag (TAMPxF) in the RTC_ISR register	To clear the flag, write zero in the TAMPxF bit.

1.8.3 Timestamp on tamper detection event

By setting the TAMPTS bit to 1, any tamper event (with edge or level detection) causes a timestamp to occur. Consequently, the timestamp flag and timestamp overflow flag are set when the tamper flag is set and work in the same manner as when a normal timestamp event occurs.

Note: It is not necessary to enable or disable the timestamp function when using this feature.

1.9 Backup registers

The 32-bit backup registers (RTC_BKPxR) are reset when a tamper detection event occurs. These registers are powered-on by VBAT when VDD is switched off (except for STM32L0 Series and STM32L1 Series), they are not reset by a system reset, and their contents remain valid when the device operates in low-power mode.

For STM32L0 Series, STM32L4 Series and STM32F7 Series, the 32-bit backup registers (RTC_BKPxR) are not reset if the TAMPxNOERASE bit is set, or if TAMPxMF (Tamper Mask Flag) is set in the RTC_TAMPCR register. Disabling the backup register reset feature allows to trigger a LPTIM event from the tamper pin without erasing the backup registers.This also allows to take benefit of the tamper input digital filtering, either to generate triggers or to generate interrupts.

Note: The VBAT availability and the LPTIM availability depend on the number of backup registers depends on the product. Refer to Table 14: Advanced RTC features.

1.10 Alternate function RTC outputs

The RTC peripheral has two outputs:

- RTC_CALIB, used to generate an external clock.
- RTC_ALARM, a unique output resulting from the multiplexing of the RTC alarm and wakeup events.

1.10.1 RTC_CALIB output

The RTC_CALIB output can be used to generate a 1 Hz or 512 Hz signal and used to measure the deviation of the RTC clock when compared to a more precise clock.

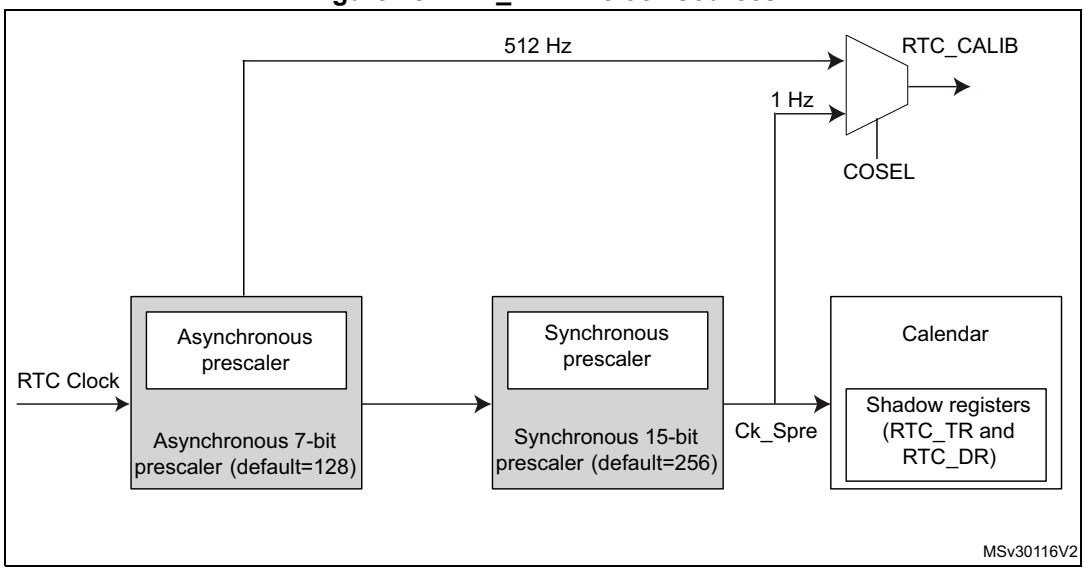
Setting 512 Hz as the output signal

1. Select LSE at 32768 Hz as RTC clock source.
2. Set the asynchronous prescaler to the default value “128“.
3. Enable the output calibration by setting “COE” to ‘1’.
4. Select 512 Hz as the calibration output by setting COSEL to ‘0’.

Setting 1 Hz as the output signal

1. Select LSE at 32768 Hz as the RTC clock source.
2. Set the asynchronous prescaler to the default value “128“.
3. Set the synchronous prescaler to the default value “256“.
4. Enable the output calibration by setting “COE” to ‘1’.
5. Select 1 Hz as the calibration output by setting COSEL to ‘1’.

Figure 18. RTC_CALIB clock sources



Maximum and minimum RTC_CALIB 512 Hz output frequency

The RTC_CALIB output can also be used to generate a variable-frequency signal. Depending on the user application, this signal can play the role of a source clock for an external device, or be connected to a buzzer to generate sound.

The signal frequency is configured using the 6 LSB bits (PREDIV_A [5:0]) of the asynchronous prescaler PREDIV_A[6:0].

Table 13. RTC_CALIB output frequency versus clock source

RTC clock source	RTC_CALIB output frequency	
	Minimum (PREDIV_A[5:0] = 111 111b) (div64)	Maximum (PREDIV_A[5:0] = 100 000b(1)) (div32)
HSE_RTC = 1 MHz	15,625 kHz	30.303 kHz
LSE = 32768 Hz	512 Hz (default output frequency)	993 Hz
LSI = 32 kHz	500 Hz	969 Hz
LSI = 37 kHz	578 Hz	1.121 kHz
LSI = 40 kHz	625 Hz	1.212 kHz

1. PREDIV_A[5] must be set to '1' to enable the RTC_CALIB output signal generation. If PREDIV_A[5] bit is zero, no signal is output on RTC_CALIB.

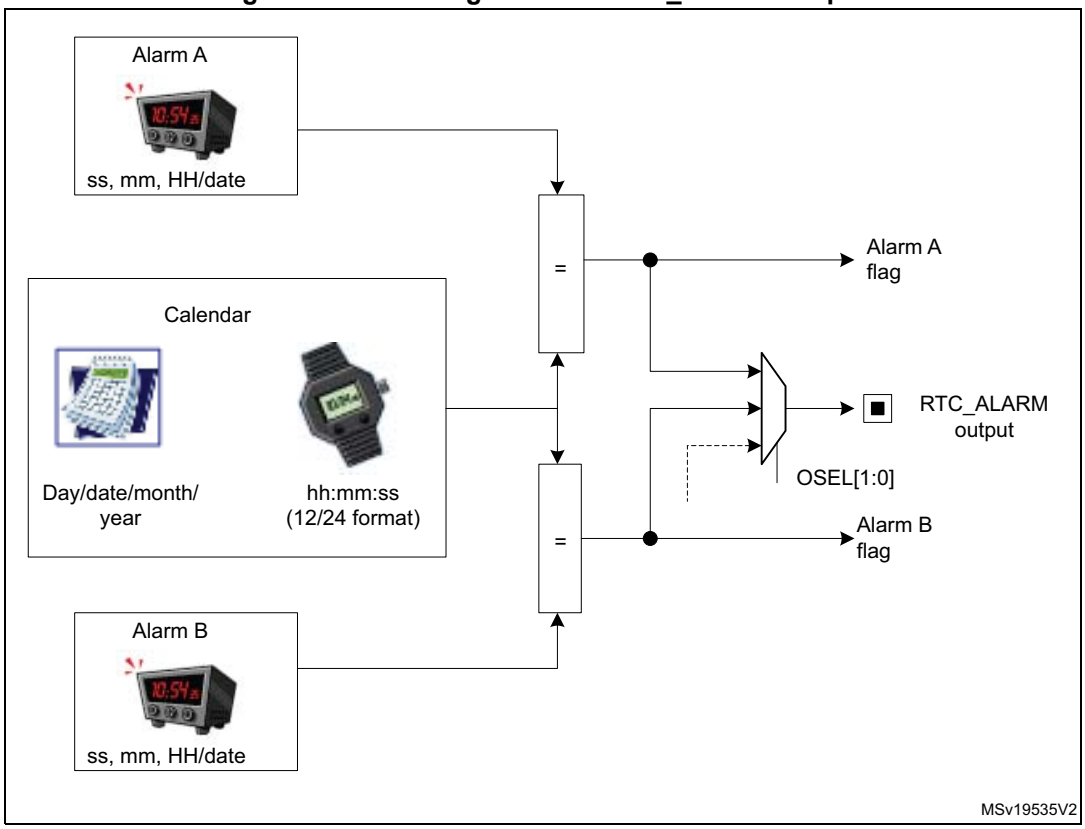
1.10.2 RTC_ALARM output

The RTC_ALARM output can be connected to the RTC alarm unit A or B to trigger an external action, or connected to the RTC wakeup unit to wake up an external device.

RTC_ALARM output connected to an RTC alarm unit

When the calendar reaches the alarm A pre-programmed value in the RTC_ALRMAR register (TC_ALRMBR register for alarm B), the alarm flag ALRAF bit (ALRBF bit), in the RTC_ISR register, is set to '1'. If the alarm A or alarm B flag is routed to the RTC_ALARM output (RTC_CR_OSEL[1:0] ="01" for alarm A, and RTC_CR_OSEL[1:0] ="10" for alarm B), this pin is set to high level or to low level, depending on the polarity selected. The output toggles when the selected alarm flag is cleared.

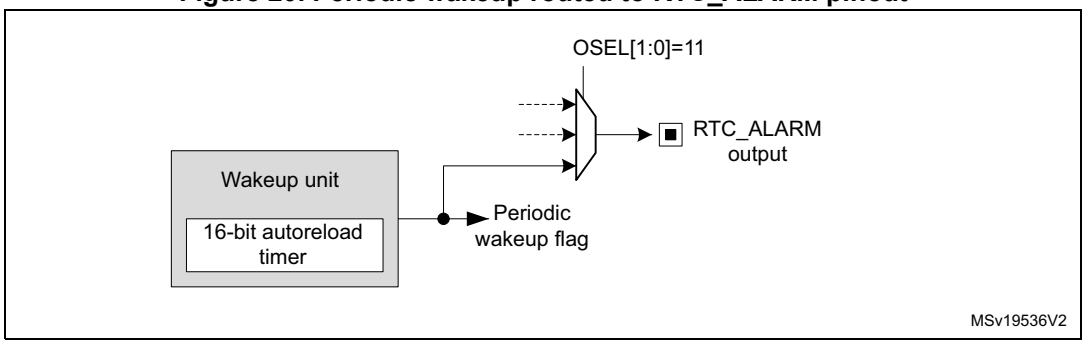
Figure 19. Alarm flag routed to RTC_ALARM output



RTC_ALARM output connected to the wakeup unit

When the wakeup downcounting timer reaches 0, the wakeup flag is set to ‘1’. If this flag is selected as the source for the RTC_ALARM output (OSEL[1:0] bits set to ‘11’ in RTC_CR register), the output will be set depending on the polarity selected and will remain set as long as the flag is not cleared.

Figure 20. Periodic wakeup routed to RTC_ALARM pinout



1.11 RTC security aspects

1.11.1 RTC register write protection

To protect the RTC registers against possible unintentional write accesses after reset, the RTC registers are initially, after a backup domain reset, locked. They must be unlocked to update the current calendar time and date.

The write-access to the RTC registers is enabled by writing a key in the Write protection register (RTC_WPR).

The following sequence is required to unlock the write protection of the RTC register:

1. Write **0xCA** into the RTC_WPR register.
2. Write **0x53** into the RTC_WPR register.

Any write access to the RTC_WPR register different from the above described write sequence activates the write-protection mechanism for the RTC registers.

1.11.2 Enter/exit initialization mode

The RTC can operate in two modes:

- *Initialization mode,* where the counters are stopped.
- *Free-running mode*, where the counters are running.

The calendar cannot be updated while the counters are running. The RTC must consequently be switched to the *Initialization mode* before updating the time and date.

When operating in this mode, the counters are stopped. They start counting from the new value when the RTC enters the *Free-running mode*.

The INIT bit of the RTC_ISR register enables the user to switch from one mode to another, and the INITF bit can be used to check the RTC current mode.

The RTC must be in *Initialization mode* to program the time and date registers (RTC_TR and RTC_DR) and the prescalers register (RTC_PRER). This is done by setting the INIT bit and waiting until the RTC_ISR_INITF flag is set.

To return to the *Free-running mode* and restart counting, the RTC must exit the *Initialization mode.* This is done by resetting the INIT bit.

Only a power-on reset can reset the calendar on microcontrollers without the VBAT pin. A system reset does not affect it but resets the shadow registers (the APB bus interface registers clocked by the APB bus clock) that are read by the application. They are updated again when the RSF bit is set. After a system reset, the application can check the INITS status flag in the RTC_ISR register to verify if the calendar is already initialized. This flag is reset when the calendar year field is set to 0x00 (power-on reset value), meaning that the calendar must be initialized. The calendar can also be reseted by setting the BDRST bit in the RCC block, even when VBAT is present.

1.11.3 RTC clock synchronization

When the application reads the calendar, it accesses shadow registers that contain a copy of the real calendar time and date clocked by the RTC clock (RTCCLK). The RSF bit is set in the RTC_ISR register each time the calendar time and date shadow registers are updated with the real calendar value. The copy is performed every two RTCCLK cycles, synchronized with the APB bus clock (the APB bus by which the RTC peripheral is

accessed). After a system reset or after exiting the initialization mode, the application must wait for the RSF bit to be set before reading the calendar shadow registers.

When the system is woken up from low-power modes (SYSCLK was off, consequently, the APB clock was off too), the application must first clear the RSF bit, and then wait until it is set again before reading the calendar registers. This ensures that the value read by the application is the current calendar value, and not the value before entering the low-power mode.

By setting the “BYPSHAD” bit to ‘1’ in the RTC_CR register, the calendar values are taken directly from the calendar counters instead of reading the shadow register. In this case, it is not mandatory to wait for the synchronization time, but the calendar registers consistency must be checked by the software. The user must read the required calendar field values. The read operation must then be performed again. The results of the two read sequences are then compared. If the results match, the read result is correct. If they do not match, the fields must be read one more time, and the third read result is valid.

Note: After resetting the BYPSHAD *bit, the shadow registers may be incorrect until the next synchronization. In this case, the software should clear the “RSF” bit then wait for the synchronization (“RSF” should be set) and finally read the shadow registers.*

2 Advanced RTC features

Table 14. Advanced RTC features

RTC features			STM32L0 Series	STM32L1 Cat. 2/3/4/5/6	STM32L1 Cat. 1	STM32L4 Series	STM32F0 Series	STM32F2 Series	STM32F3 Series	STM32F4 Series	STM32F7 Series
RTC clock source (LSE, LSI, HSE with prescaler)			X	X	X	X	X	X	X	X	X
Prescalers	Asynchronous		X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)	X (7 bits)
	Synchronous		X (15 bits)	X (15 bits)	X (13 bits)	X (1 5 bits)	X (15 bits)	X (13 bits)	X (15 bits)	X (15 bits)	X (15 bits)
Calendar	Time	12/24 format	X	X	X	X	X	X	X	X	X
		Hour, minutes and seconds	X	X	X	X	X	X	X	X	X
		Sub-second	X	X	Not available	X	X	Not available	X	X	X
	Date		X	X	X	X	X	X	X	X	X
	Daylight operation		X	X	X	X	X	X	X	X	X
	Bypass the shadow registers		X	X	Not available	X	X	Not available	X	X	X
	VBAT mode		Not available	Not available	Not available	X	X	X	X	X	X

Table 14. Advanced RTC features (continued)

RTC features			STM32L0 Series	STM32L1 Cat. 2/3/4/5/6	STM32L1 Cat. 1	STM32L4 Series	STM32F0 Series	STM32F2 Series	STM32F3 Series	STM32F4 Series	STM32F7 Series
Alarm	Alarms available	Alarm A	X	X	X	X	X	X	X	X	X
		Alarm B	X	X	X	X	Not available	X	X	X	X
	Time	12/24 format	X	X	X	X	X	X	X	X	X
		Hour, minutes and seconds	X	X	X	X	X	X	X	X	X
		Sub-second	X	X	Not available	X	X	Not available	X	X	X
	Date or week day		X	X	X	X	X	X	X	X	X
Tamper detection	Configurable input mapping		X	Not available	Not available	X	X	X	X	X	X
	Configurable edge detection		X	X	X	X	X	X	X	X	X
	Configurable Level detection (filtering, sampling and precharge configuration on tamper input)		X	X	Not available	X	X	Not available	X	X	X
	Number of tamper inputs		3 inputs / 3 events	3 inputs / 3 events	1 input / 1 event	3 inputs / 3 events	2 inputs 3 inputs (for STM32F07x and STM32F09x)	2 inputs / 1 event	2 inputs / 2 events	2 inputs / 2 events	3 inputs/ 3 events
	VBAT mode pins		Not available	Not available	Not available	3 inputs	1 input	1 input	1 input	2 inputs	2 inputs

Table 14. Advanced RTC features (continued)

RTC features			STM32L0 Series	STM32L1 Cat. 2/3/4/5/6	STM32L1 Cat. 1	STM32L4 Series	STM32F0 Series	STM32F2 Series	STM32F3 Series	STM32F4 Series	STM32F7 Series
Time Stamp	Configurable input mapping		X(1)	Not available	Not available	X(1)	X	X	X	X	X
	Time	Hours, minutes and seconds	X	X	X	X	X	X	X	X	X
		Sub-seconds	X	X	Not available	X	X	Not available	X	X	X
	Date (Day, Month)		X	X	X	X	X	X	X	X	X
	Time Stamp on tamper detection event		X	X	X	X	X	X	X	X	X
	Time Stamp on switch to VBAT mode		Not available	Not available	Not available	X	Not available	Not available	Not available	Not available	X
RTC Outputs	RTC_ ALARM	Alarm event	X	X	X	X	X	X	X	X	X
		Wakeup event	X	X	X	X	X	X	X	X	X
	RTC_ CALIB	512 Hz	X	X	X	X	X	X	X	X	X
		1 Hz	X	X	Not available	X	X	Not available	X	X	X
RTC Calibration	Coarse Calibration		Not available(2)	X	X	Not available(2)	Not available(2)	X	X	X	Not available(2)
	Smooth Calibration		X	X	Not available	X	X	Not available	X	X	X
Synchronizing the RTC			X	X	Not available	X	X	Not available	X	X	X
Reference clock detection			X	X	X	X	X	X	X	X	X

Table 14. Advanced RTC features (continued)

RTC features		STM32L0 Series	STM32L1 Cat. 2/3/4/5/6	STM32L1 Cat. 1	STM32L4 Series	STM32F0 Series	STM32F2 Series	STM32F3 Series	STM32F4 Series	STM32F7 Series
Backup registers	Powered-on VBAT	Not available	Not available	Not available	X	X	X	X	X	X
	Reset on a tamper detection	X	X	X	X	X	X	X	X	X
	Reset when Flash readout protection is disabled	X	X	X	X	X	Not available	X	Not available	X
	Number of backup registers	5	32(3)	20(4)	32	5	20	16	20	32

1. Thanks to timestamp on tamper event.
2. Obsolete, replaced by smooth calibration.
3. Cat. 3/4/5/6.
4. Cat. 1 and Cat. 2.

3 Reducing power consumption

The RTC is designed to minimize the power consumption: the Real-Time Clock functionality only adds typically 300nA to the current consumption.

The RTC keeps working in reset mode and its registers are only reset by a VBAT power-on, if it has previously been powered off. As the RTC register values are not lost after a system reset, the calendar keeps the correct time and date until VDD and VBAT power down.

Note: In the STM32L0 Series and STM32L1 Series, the VBAT mode is not available. When VBAT is present, it can anyway be connected to VDD. In these cases, consider replacing VBAT by VDD in the previous sentence.

3.1 Using the right power reduction mode

Depending on the application constraints, such as the maximum or average current consumption, the frequency of wake-ups, or alternatively the maximum wake-up time, several low-power modes can be used.

The RTC peripheral can be active in the following low-power modes:

- Sleep mode
- Low-power Run mode
- Low-power Sleep mode
- Stop mode if the RTC clock is provided by LSE or LSI[a]
- Standby mode if the RTC clock is provided by LSE or LSI
- Shutdown mode if the RTC clock is provided by LSE[b]

Note: The low-power modes are only present on STM32L0 Series and STM32L4 Series. The shutdown is only present on STM32L4 Series.

3.2 Use tamper pin internal pull-up resistor

For all STM32 MCUs (except STM32L1 Cat 1. and STM32F2 Series), the internal pull-up resistor in the Tamper Input pad is only applied for a short period of time (1, 2, 4 or 8 RTCCLK cycles) so using the RTC internal pull-up resistor instead of standard IO pull-up/pull-down or external pull-up/pull-down ensures the lowest power consumption.

The trade-off between the tamper detection latency and the power consumption by the RTC internal pull-up can be optimized using TAMPFREQ field. It determines the frequency of the tamper sampling from 128 Hz to 1 Hz when RTCCLK equals 32768 Hz.

a. In the STM32L4 Series, there are 3 Stop modes. In order to select the most relevant mode, refer to [11] and [12] for the other series.

b. Only for the STM32L4 Series.

3.3 Setting the RTC prescalers

The prescalers used for the calendar are divided into a asynchronous and a synchronous prescalers. Increasing the value of the asynchronous prescaler (and reducing the synchronous prescaler accordingly to keep the 1 Hz output) reduces the RTC power consumption.

4 STM32L4 API and tampering detection application example

4.1 STM32 Cube firmware libraries

The Real-Time Clock peripheral comes with:

- A firmware driver API abstracting the RTC features for the end-user. Refer to stm32l4xx_hal_rtc.c and stm32l4xx_hal_rtc_ex.c files in \STM32Cube_FW_L4_Vx.y.z\Drivers\STM32L4xx_HAL_Driver\
- A set of example projects so that the user can quickly become familiar with the RTC peripheral. Refer to the examples in \STM32Cube_FW_L4_Vx.y.z\Projects\STM32L476RG-Eval\Examples\RTC

4.2 STM32 Cube expansion firmware

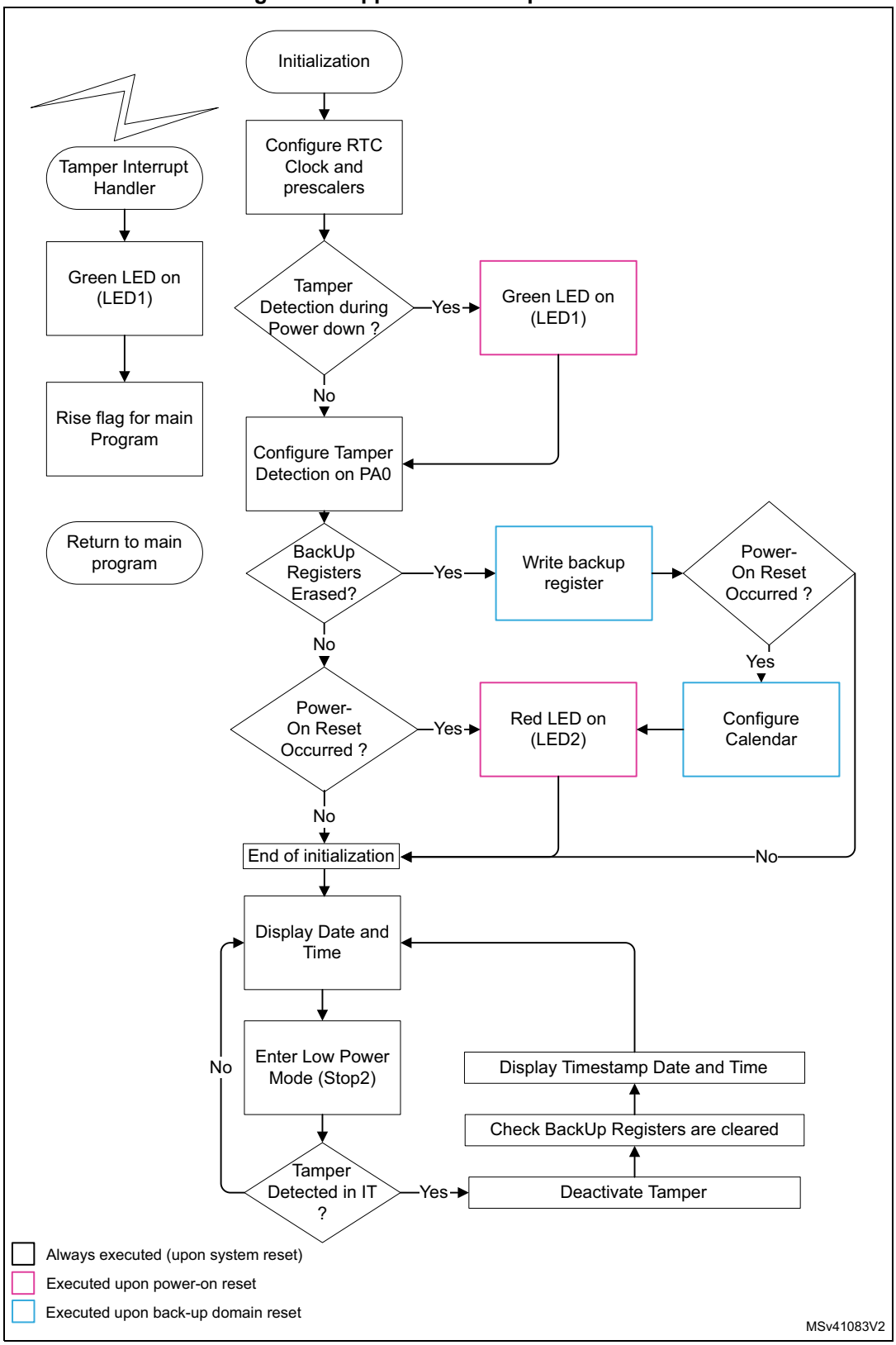
This application note comes with the X-CUBE-RTC expansion software upon the STM32 Cube firmware libraries.

It shows a concrete example where a real-time clock must be maintained alive while using as less power as possible. The firmware implements:

- A few hints order to ensure a low current consumption,
- A display in the debugger (date and time, timestamp date and time) and on LEDs (power up events, reset events, tamper events, error)
- A display on the TFT screen, using STemWin library
- The tampering detection capability both when the main power supply (VDD) is present and when the RTC is battery powered,
- The ability to timestamp the tampering event,
- The ability to erase the backup registers that may contain sensitive data upon tamper detection.

The application flowchart is shown in *Figure 21*

Figure 21. Application example flowchart

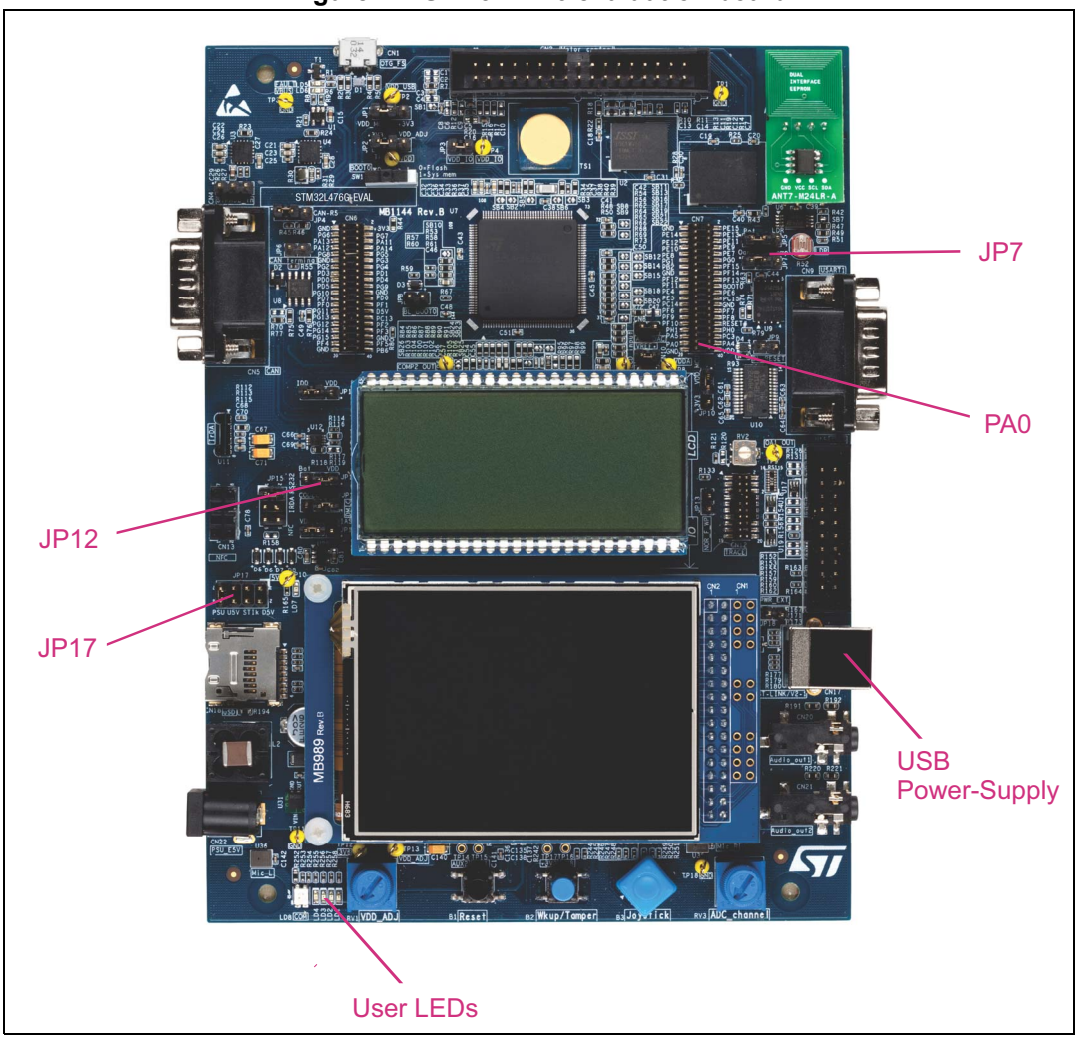


4.3 Application example project

4.3.1 Hardware Setup

In order to use the application example project, the user needs a STM32L476 evaluation board (order code: STM32L476G-EVAL).

Figure 22. STM32L476 evaluation board



Supply the board using the USB cable. For this example to work, the user needs to tie PA0 (tamper pin 2) to 0. In order to do that :

- Remove the jumper JP7
- Connect the CN7 pin 37 to GND.

Note: PC13 (tamper pin 1) connected to the blue push-button is not used in this demonstration firmware as it is connected to an external pull-down resistor preventing the use of the tamper detection on level with internal pull-up (lowest consumption mode).

For more information, refer to [9].

4.3.2 Software setup

Open the project under the user's favorite integrated development environment. For IAR EWARM, open the «project.eww» file. For Keil MDK-ARM, open the «project.uvprojx» file, compile and launch the debug session. For SW4STM32, open the SW4STM32 toolchain, browse to the SW4STM32 workspace directory, select and import the project RTC_TamperVBATT. Launching the debug session will load the program in the internal Flash memory and execute it.

In EWARM debug session, the user can view the calendar and tamper event timestamp (time and date) by opening a «Live watch» window (View -> Live watch) and observe the following global variables:

- aShowTime
- aShowDate
- aShowTimeStampTime
- aShowTimeStampDate

The same observation can be done using Keil/MDK-ARM development tools. To open "Watch1" window, do View->Watch Windows -> Watch1.

4.3.3 LED meaning

LD1 (Green): Tamper event detected

TFT displays: the Tamper date and the Tamper time are different from their initialization state

LD2 (Orange): Power-On Reset occurred

TFT displays: the Tamper date and the Tamper time are at their initialization state

LD3 (Red): Error (RTC or RCC configuration error, backup registers not erased)

TFT displays: “error occurred” is displayed

LD4 (Blue): Reset occurred.

TFT displays: the Tamper date and the Tamper time are at their initialization state

4.3.4 Tampering detection during normal operation

Disconnect the debugger. A Power-On Reset can be generated by removing the JP17 jumper and placing it again on STlk.

Open PA0 (input is now floating), the LD1 lits showing that a tamper event is detected. On TFT, the Tampering date and Tampering time will be updated with the timestamp of the Tamper event. Note that the user observes a long delay (up to 2 seconds) before the tamper event is detected. This is due to the trade-off in the tamper detection frequency (TAMPFREQ) chosen which ensures the lowest power consumption.

A reset can be applied by pushing the black reset button. The application is now able the detect a new tamper event.

4.3.5 Tampering detection when main power supply is off

If the user supplies the VBAT pin with the external CR1220 battery (JP12 on BAT position), the tamper event can be detected even if the main power supply is off. Proceed as follow:

- Remove the JP17 jumper
- Open PA0
- Tie PA0 to GND (this is simulating that the end-user takes care to close the box containing the electronics before supplying it again)
- Set JP17 on STlk
- The LD1 lits showing that a tamper event has been detected during the power down and on TFT, the Tampering date and Tampering time will be updated with the timestamp of the Tamper event

Note that if the user supplies the VBAT pin with 3V (JP12 on VDD position), the RTC is no longer supplied when the main power supply is switched off. In this case, the application is no longer able to detect the tampering event while the main power supply is off.

5 STM32L4 API and digital smooth calibration application example

5.1 STM32 Cube firmware libraries

The TIMER peripheral comes with:

- A firmware driver API abstracting the TIMER features for the end-user. Refer to stm32l4xx_hal_tim.c and stm32l4xx_hal_tim_ex.c files in \STM32Cube_FW_L4_Vx.y.z\Drivers\STM32L4xx_HAL_Driver\
- A set of example projects so that the user can quickly become familiar with the TIMER peripheral. Refer to the examples in \STM32Cube_FW_L4_Vx.y.z\Projects\STM32L476RG-Nucleo\Examples\TIM

The Real-Time Clock peripheral comes with:

- A firmware driver API abstracting the RTC features for the end-user. Refer to stm32l4xx_hal_rtc.c and stm32l4xx_hal_rtc_ex.c files in \STM32Cube_FW_L4_Vx.y.z\Drivers\STM32L4xx_HAL_Driver\
- A set of example projects so that the user can quickly become familiar with the RTC peripheral. Refer to the examples in \STM32Cube_FW_L4_Vx.y.z\Projects\STM32L476RG-Nucleo\Examples\RTC

5.2 STM32 Cube expansion firmware

This application note comes with the X-CUBE-RTC expansion software upon the STM32 Cube firmware libraries.

This example shows an implementation of the digital smooth calibration feature of the RTC. It also uses the GPTIMER HAL API.

It shows a concrete example where RTC_CLK is smoothly calibrated based on an external 1 Hz reference. The firmware implements:

- A GP timer in Master mode using channel 1 in output compare mode. It uses an external trigger clock.
- A GP timer in Master mode using channel 1 in input capture mode and channel 2 in output compare mode. It is also configured to use the LSE clock thanks to the TIM option register.
- The RTC with the programmation of the CALR register.
- GPIOs to connect the input 1 Hz reference clock and the ‘corrected’ RTC_OUT_CALIB clock.

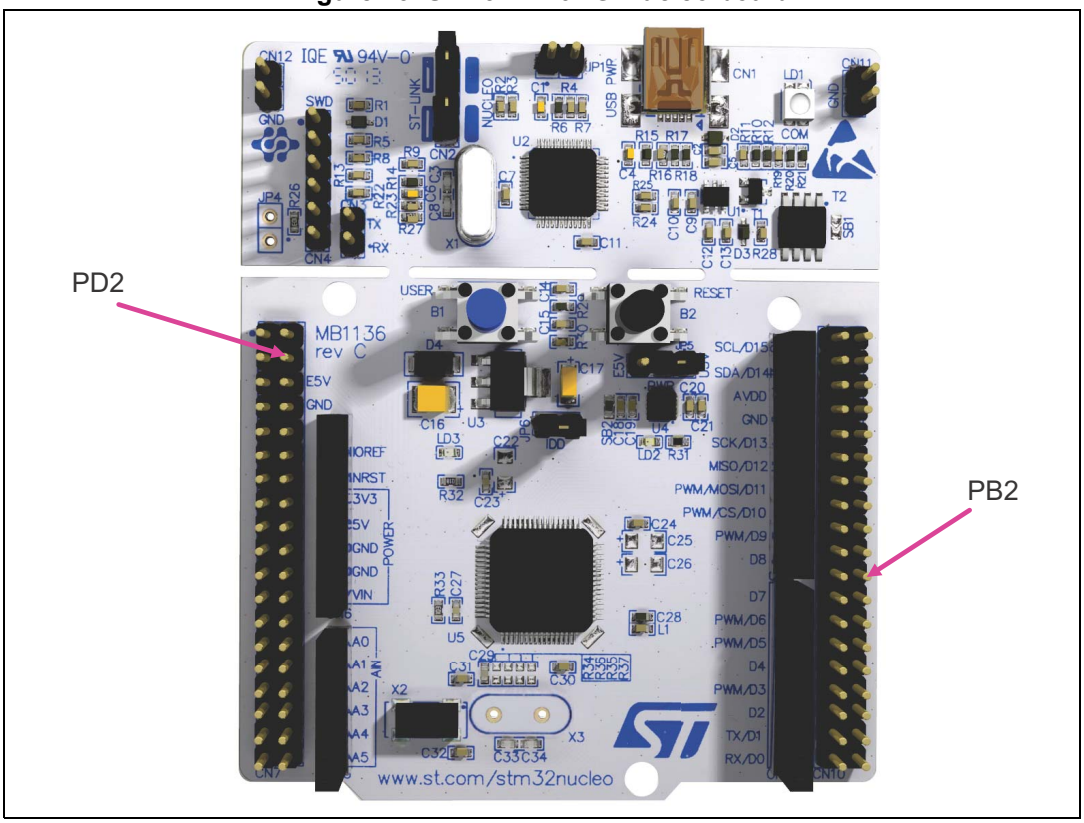
The application block diagram is shown in *Figure 23*.

5.3 Application example project

5.3.1 Hardware setup

In order to use the application example project, the user needs a STM32L476RG Nucleo board.

Figure 23. STM32L476RG-Nucleo board



Supply the board using the USB cable. For this example, the user needs to:

- Connect PD2 to a signal generator driving 1 Hz clock with 3.3 V amplitude. It is advised to control the generated frequency at few ppm.
- Connect PB2 to an oscilloscope. Note that the accuracy of the frequency measurement is ideally in the range of 1 ppm.

5.3.2 Software setup

Open the project under the user's favorite integrated development environment. For IAR EWARM, open the "RTC_SmoothCalib.eww" file. For Keil MDK-ARM, open the "RTC_SmoothCalib.uvprojx" file. For SW4STM32, open the SW4STM32 toolchain, browse to the SW4STM32 workspace directory, select and import the project RTC_SmoothCalib.

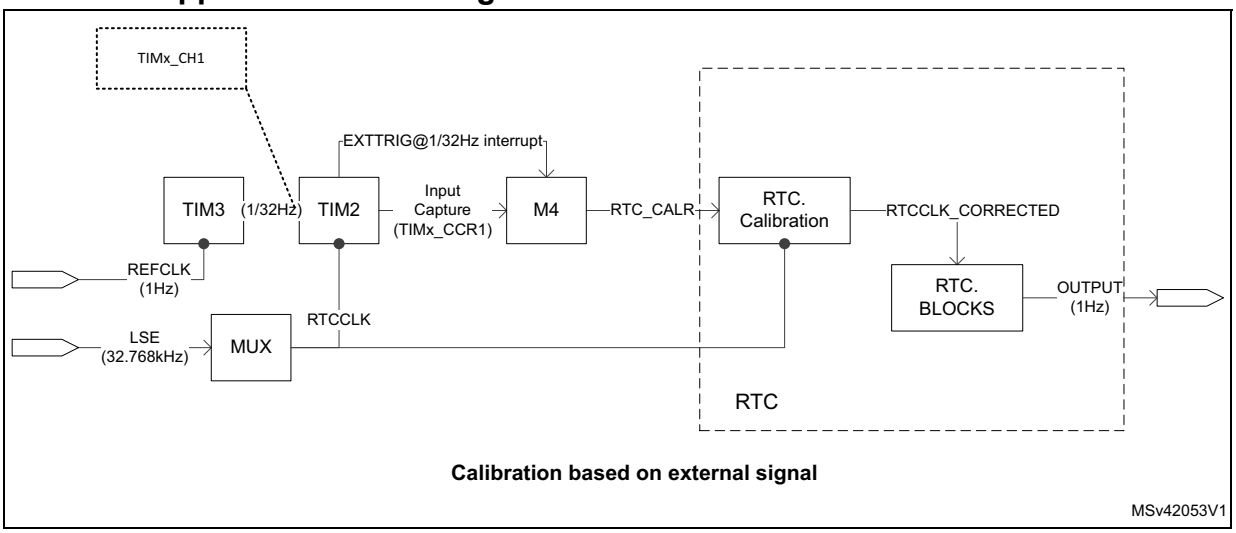
Compile and launch the debug session. It loads the program in the internal Flash memory and executes it.

After a maximum of 64 seconds, make a frequency measurement of the CALIB_OUT clock on PB2. The user shall get a frequency accurate at 1 ppm compared to the 1 Hz reference clock sent on PD2.

The user can then modify the PD2 clock by few ppm, wait 64 seconds and check that the PB2 clock has been changed accordingly.

Under debugger, the user can monitor the evolution of the CALR register fields: CALP and CALM.

5.3.3 Application block diagram



The application is based on 4 steps:

1. A 1Hz reference clock is connected to TIM3 through the PD2 GPIO
2. TIM3 is configured to generate a rising edge after 32 x 1 Hz reference rising edges. This gives an event every 32 seconds.
3. TIM2 is configured to increment its counter on CLK_RTC and to get the counter value on the rising edge generated by TIM3 and stores it in the TIM2_CCR1 register. It then reset itslef.
4. The cortex-M4 core gets the TIM2_CCR1 value, compares it with the number of CLK_RTC cycles expected in 32 seconds and processes the comparison results to update the CALP and CALM fields of the RTC_CALR register.

The calibration is continous.

5.3.4 Run time observations

To check the correct functionality:

1. Modify the 1 Hz reference clock, for instance by forcing 1.0001 Hz
2. Wait 2 x 32 seconds
3. Measure the output frequency. Depending of the accuracy of the generator and the measurement device used, the user can see around 1 ppm accuracy.

The user can also monitor in debug mode:

- The RTC_CALR register (CALM and CALP fields)
- The TIM2_CCR1 register (which contains the number of CLK_RTC cycles within a 32 second window)

6 STM32L0 API and tampering detection application example

6.1 STM32 Cube firmware libraries

The Real-Time Clock peripheral comes with:

- A firmware driver API abstracting RTC features for the end-user. Refer to stm32l0xx_hal_rtc.c and stm32l0xx_hal_rtc_ex.c files in \STM32Cube_FW_L0_Vx.y.z\Drivers\STM32L0xx_HAL_Driver\
- A set of example projects so that the user can quickly become familiar with the RTC peripheral. Refer to the examples in \STM32Cube_FW_L0_Vx.y.z\Projects\STM32L053R8-Nucleo\Examples\RTC

6.2 STM32 Cube expansion firmware

This application note comes with the X-CUBE-RTC expansion software upon the STM32 Cube firmware libraries.

It shows a concrete example where a real-time clock must be maintained alive while using as less power as possible. The firmware implements:

- A few hints order to ensure a low current consumption,
- A display in the debugger (date and time, timestamp date and time) and on a single LED (power up events, reset events, tamper events, error)
- The tampering detection capability,
- The ability to timestamp the tampering event,
- The ability to erase the backup registers that may contain sensitive data upon tamper detection.

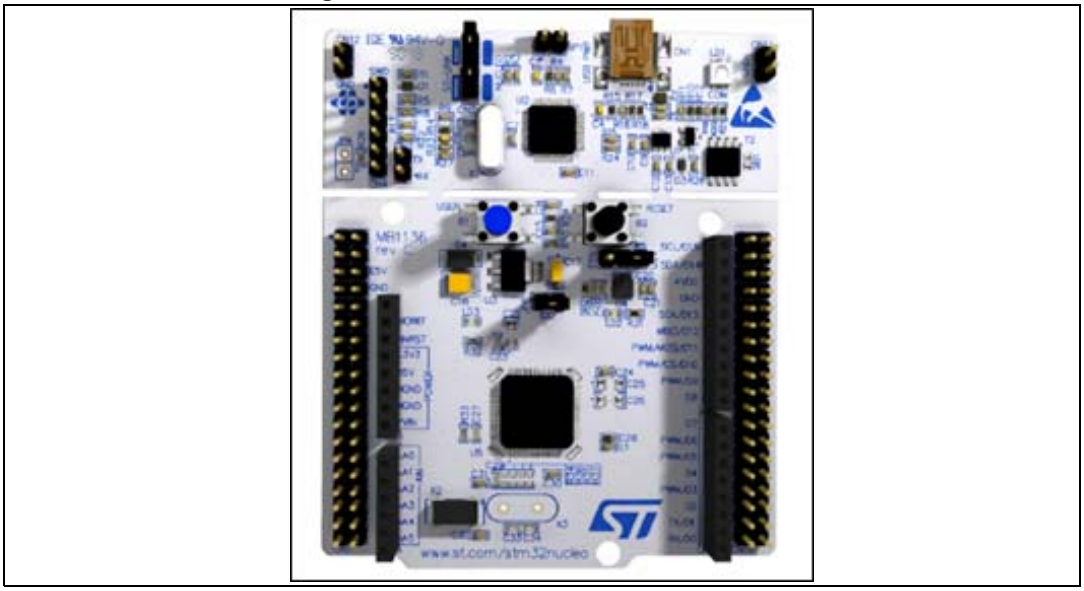
The application flowchart is shown in *Figure 21: Application example flowchart*.

6.3 Application example project

6.3.1 Hardware setup

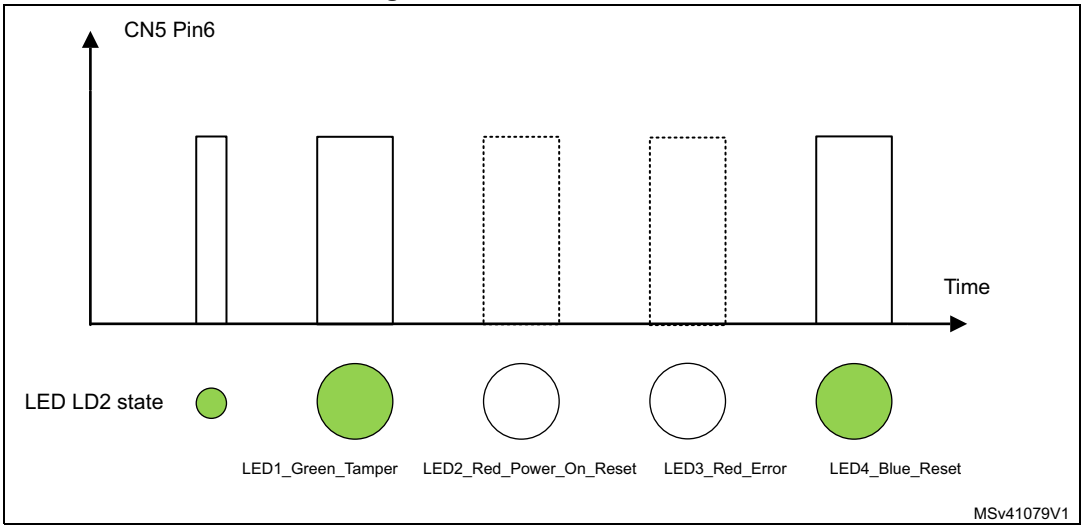
In order to use the application example project, the user needs a STM32L053R8-Nucleo board (order code STM32L053R8-NUCLEO).

Figure 24. STM32L053R8-Nucleo board



Supply the board using the USB cable. In this example, the user uses the B1 push button to simulate the external tamper event. The user can observe the software "private variables" on LED LD2 and CN5 Pin 6 (SCK/D13). Details regarding the LED LD2 behavior are available in main.c file.

Figure 25. LED LD2 behavior



6.3.2 Software setup

Open the project under the user's favorite integrated development environment. For IAR EWARM, open the "project.eww" file. For Keil MDK-ARM, open the "project.uvprojx" file. For SW4STM32, open the SW4STM32 toolchain, browse to the SW4STM32 workspace directory, select and import the project RTC_Tamper.

Compile and launch the debug session. This will load the program in the internal Flash memory and execute it.

In EWARM debug session, the user can view the calendar and tamper event timestamp (time and date) by opening a "Live watch" window (View -> Live Watch) and observe the following global variables:

- aShowTime
- aShowDate
- aShowTimeStampTime
- aShowTimeStampDate

The user can add and follow these "private variables":

- LED1_Green_Tamper_event_detected
- LED2_Red_Power_On_Reset_occurred
- LED3_Red_Error
- LED4_Blue_Reset_occurred

The same observation can be done using Keil/MDK-ARM development tools. To open "Watch1" window, do View->Watch Windows -> Watch1.

6.3.3 LED meaning.

Only one LED (LD2) is available on the STM32L053R8-Nucleo board. To match with the STM32L4 examples, described in *Section 4: STM32L4 API and tampering detection application example*, LED1, LED2, LED3 and LED4 are replaced by 4 integers:

- uint32_t LED1_Green_Tamper_event_detected:
  Set when tamper1 event is detected.
  Equivalent to LED1_Green in STM32L4 examples.
- uint32_t LED2_Red_Power_On_Reset_occurred:
  Set when power-on reset is detected.
  Equivalent to LED2_Red in STM32L4 examples.
- uint32_t LED3_Red_Error:
  Set in errors cases.
  Equivalent to LED3_Red in STM32L4 examples.
- uint32_t LED4_Blue_Reset_occurred:
  Set when reset is detected (B2 black push button).
  Equivalent to LED4_Blue in STM32L4 examples.

The user can observe these "private variables" using the live watch feature in the debugger.

The user can observe these "private variables" on LED LD2 and CN5 Pin 6 (SCK/D13). Regarding the LED LD2 behavior, see details in main.c file.

6.3.4 Tampering detection during normal operation

Step1: a Power-On-Reset can be generated by disconnecting/connecting the USB cable on CN1.

Table 15. Tampering detection status when Power-On-Reset is detected

Meaning	Status	LED LD2 (Blink)
LED1_Green_Tamper_event_detected	0	OFF
LED2_Red_Power_On_Reset_occurred	1	ON
LED3_Red_Error	0	OFF
LED4_Blue_Reset_occurred	1	ON

Step2: Push B1 to simulate a tamper event.

Table 16. Tampering detection status when tamper event is detected

Meaning	Status	LED LD2 (Blink)
LED1_Green_Tamper_event_detected	1	ON
LED2_Red_Power_On_Reset_occurred	1	ON
LED3_Red_Error	0	OFF
LED4_Blue_Reset_occurred	1	ON

Step3: Push B2 to simulate a reset.

Table 17. Tampering detection status when reset is detected

Meaning	Status	LED LD2 (Blink)
LED1_Green_Tamper_event_detected	0	OFF
LED2_Red_Power_On_Reset_occurred	0	OFF
LED3_Red_Error	0	OFF
LED4_Blue_Reset_occurred	1	ON

Step4: Push B1 to simulate a tamper event.

Table 18. Tampering detection status when tamper event is detected

Meaning	Status	LED LD2 (Blink)
LED1_Green_Tamper_event_detected	1	ON
LED2_Red_Power_On_Reset_occurred	0	OFF
LED3_Red_Error	0	OFF
LED4_Blue_Reset_occurred	1	ON

7 Reference documentation

[1]. *STM32L4x6 advanced ARM®-based 32-bit MCUs* reference manual (RM0351)

[2]. *Ultra-low-power STM32L0xx advanced ARM®-based 32-bit MCUs* reference manuals (RM0367, RM0376, RM0377)

[3]. *STM32L100xx, STM32L151xx, STM32L152xx and STM32L162xx advanced ARM®-based 32-bit MCUs* (RM0038)

[4]. *STM32F0x1/STM32F0x2/STM32F0x8 advanced ARM®-based 32-bit MCUs* reference manual (RM0091)

[5]. *STM32F205xx, STM32F207xx, STM32F215xx and STM32F217xx advanced ARM®-based 32-bit MCUs* reference manual (RM033)

[6]. *STM32F301x6/8 and STM32F318x8 advanced ARM®-based 32-bit MCUs* reference manual (RM0366)

[7]. *STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM®-based 32-bit MCUs* reference manual (RM090)

[8]. *STM32F75xxx and STM32F74xxx advanced ARM®-based 32-bit MCUs* reference manual (RM0385)

[9]. *Evaluation board with STM32L476ZGT6 MCU* user manual (UM1855)

[10]. *STM32 Nucleo-64 boards* user manual (UM1724)

[11]. *Optimizing power and performances with STM32L4 Series microcontrollers* application note (AN4746)

[12]. *STM32L0xx ultra-low power features overview* application note (AN4445)

8 Revision history

Table 19. Document revision history

Date	Revision	Changes
26-May-2016	1	Initial release.
25-Oct-2016	2	– Updated *Table 1: Applicable products* adding product series and part number. – Updated *Figure 3: STM32L0 Series, STM32L1 Series, STM32F2 Series and STM32F4 Series RTC clock sources*. – Updated *Figure 4: STM32L4 Series, STM32F0 Series, STM32F3 Series and STM32F7 Series RTC clock sources* – Updated *Table 3: Calendar clock equal to 1 Hz with different clock sources*. – Updated *Section 1.4: Digital smooth calibration* adding the RTC calibration basics. – Updated *Table 14: Advanced RTC features*. – Updated *Section 4.3.3: LED meaning*. – Added *Section 5: STM32L4 API and digital smooth calibration application example*. – Updated *Section 6.3.1: Hardware setup*. – Updated *Section 6.3.2: Software setup*. – Updated *Section 7: Reference documentation*.
11-May-2017	3	– Updated *Figure 4: STM32L4 Series, STM32F0 Series, STM32F3 Series and STM32F7 Series RTC clock sources*. – Updated *Table 3: Calendar clock equal to 1 Hz with different clock sources* adding note. – Updated *Section 1.4.1: RTC calibration basics*. – Updated *Section 1.11.1: RTC register write protection*. – Updated *Table 14: Advanced RTC features* RTC calibration for STM32F2 Series.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2017 STMicroelectronics – All rights reserved