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